

版权注意事项：

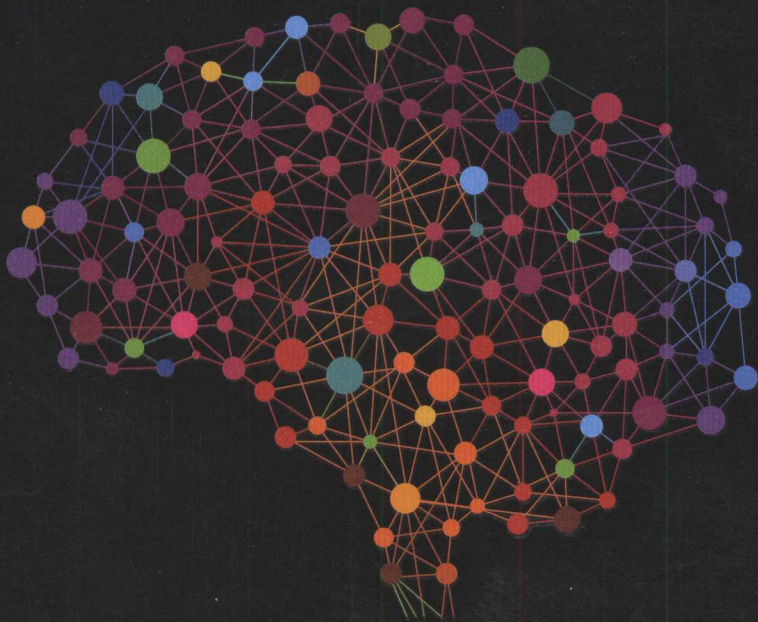
- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

NLP，让人类与智能机器的交互不再遥远；
深度学习，让语言解析不再是智能系统的瓶颈。

NLP汉语自然语言处理

原理与实践

郑捷◎著



围绕三个部分展开

自然语言理论、人工智能算法、算法实现和案例部分

多达15个算法的讲解

包含NShort、HMM、朴素贝叶斯、CRF、BP神经网络等

多达9个程序库的剖析

包括Python NLTK、Ltp3.X、HanLP、Word2Vec、CRF++、Keras等



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

作者介绍

郑捷：www.threedweb.cn网站的负责人，研究方向是机器学习与自然语言处理。当前负责的核心产品是高精度自然语言认知系统的设计与研发，研发目标是高精度（识别率为85%~95%）的统一架构的NLP认知系统，已经出版专著《机器学习算法原理与编程实践》，希望能与在NLP这方面有兴趣的读者一起学习交流。

本书核心内容

NLP中的开源系统及其应用
中文分词源码解析
概率图模型的理论与算法
使用概率图模型进行序列标注
语料库的介绍与建设
深度学习与NLP
NLP与认知理论
汉语的句法与语义的解析

读者对象

对自然语言处理感兴趣的各类人士
各类NLP入门学习者
各大人工智能实验室、软件学院等专业机构的研究者
机器学习和NLP的算法设计师
物联网设计人员
目光敏锐的创业者

NLP 汉语自然语言处理

原理与实践

郑捷◎著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内容简介

本书是一本研究汉语自然语言处理方面的基础性、综合性书籍，涉及 NLP 的语言理论、算法和工程实践的方方面面，内容繁杂。

本书包括 NLP 的语言理论部分、算法部分、案例部分，涉及汉语的发展历史、传统的句法理论、认知语言学理论。需要指出的是，本书是一本系统介绍认知语言学和算法设计相结合的中文 NLP 书籍，并从认知语言学的视角重新认识和分析了 NLP 的句法和语义相结合的数据结构。这也是本书的创新之处。

本书适用于所有想学习 NLP 的技术人员，包括各大人工智能实验室、软件学院等专业机构。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目 (CIP) 数据

NLP 汉语自然语言处理原理与实践 / 郑捷著. —北京: 电子工业出版社, 2017.1

ISBN 978-7-121-30765-2

I. ①N… II. ①郑… III. ①汉语—自然语言处理—研究 IV. ①TP391

中国版本图书馆 CIP 数据核字 (2016) 第 321878 号

策划编辑: 李 冰

责任编辑: 李 冰

特约编辑: 田学清 罗树利等

印 刷: 涿州市京南印刷厂

装 订: 涿州市京南印刷厂

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱

邮编: 100036

开 本: 787×1092 1/16 印张: 34 字数: 816 千字

版 次: 2017 年 1 月第 1 版

印 次: 2017 年 5 月第 2 次印刷

定 价: 98.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话: (010) 88254888, 88258888。

质量投诉请发邮件至 zltts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式: libing@phei.com.cn。

推荐序

自然语言处理是人工智能领域的一颗明珠，现在已经成为人工智能研究中最活跃的领域。几十年来，随着计算机技术和人工智能技术的发展，自然语言处理取得了长足的进步。现在，自然语言处理技术正处在一个新的历史转折点，随着可获取信息量的爆炸性增长，信息过载问题愈发严重，以词法分析和词义理解为主的传统自然语言处理技术已经难以满足解决实际问题的需要，句子级乃至篇章级语义理解技术即将成为人工智能技术发展的新趋势。

自然语言处理作为人工智能与语言学的综合学科，理应从两个学科中汲取营养来推动自身的进步。但目前概率和数据驱动的方法在自然语言处理领域占据绝对的主流，加之近几年深度学习的异军突起，语言学知识在自然语言处理领域中受到的重视程度愈发不足。而以我在自然语言处理领域工作的经验来看，越深入研究，越能感觉到语言学知识不足的掣肘。特别是深层次的语义理解，脱离了语言学知识，就会变成无源之水、无本之木。常见的自然语言处理书籍对于解决具体问题的方法讲解已经足够丰富，但对于语言学基础理论的介绍和思考还略显不足。一些前辈虽然一直在思考语言和认知的本质，但其发表出来的内容只限于思考结果的一鳞半爪，较少结集成书。加之现在自然语言处理领域的学习者大多是计算机背景，极少系统地学习过语言学的基础理论。这样造成的现状就是从事自然语言处理的技术人员越来越多，但相互之间的讨论和经验分享多集中于具体的技术手段或算法的数学原理，而极少涉及语言学的基础理论和语义理解的本质问题。

本书作者通过对前人语言学理论和自然语言处理技术的深入梳理，形成了自己对于语义理解，特别是汉语语义理解独特的思考和一整套理论体系，提出了语义理解的系统解决之道。尽管如何才能让计算机理解语义，在学术界还没有定论，但作者系统性的思考和解决思路是非常难能可贵的。本书在内容上保证了理论和技术的平衡，在介绍术的同时，充分展示了作者对于道的思考成果。此书是自然语言处理书籍中的一股新风，希望其可以对语义理解的研究和发展起到积极的推动作用，同时引导自然语言处理领域的研究者，特别是初学者，加强对于语言学的理论的学习，更多地从问题的本源来寻求新的解决思路，而不仅仅满足于在传统解决思路上尝试新的技术手段。

愿每一位有志于从事自然语言处理的研究者，都能从此书中获得一些启示。

贾文杰：早年在富士通研发中心，著名的 1998 年人民日报语料库的研发单位之一，任高级研究员，负责情感分析，后进入 360 搜索引擎自然语言处理部，项目核心成员之一，主持搜索引擎分词，纠错等核心模块研发工作，历时 3 年，对搜索效果的提升起到重要作用。目前，转入移动互联领域，负责猎豹移动的自然语言处理部，任负责人。

前言

写作本书的动机

自然语言处理（Natural Language Processing, NLP）是人工智能和语言学领域的分支学科，主要研究如何让计算机处理和运用自然语言。自然语言处理广义上分为两大部分，第一部分为自然语言理解，是指让电脑“懂”人类的语言；第二部分为自然语言生成，是指把计算机数据转化为自然语言。本书重点讲解汉语自然语言处理方面的最新理论、技术和进展。

自然语言处理作为一个独立的学科诞生至今，已经半个多世纪了。与绝大多数传统学科的最大不同是，在这半个世纪中，它始终离问题的终结遥遥无期，当人们千辛万苦地获得一次又一次的突破后，又会被新出现的问题无情地阻拦，而再次陷入迷惘之中。在 NLP 中，问题好像没有最终解决方案，甚至连最佳实践也没有，而只有最新现状（State of art）。而近些年，那些历史上的 State of art 正被不断地刷新、不断地超越。

就在十多年前，商业化的人机交互都是人们可望而不可即的目标，但现在智能机器人正逐渐走入市场，走入人们的生活。虽然这些技术还不够成熟，还要解决诸多问题，即便普通大众也能意识到，我们离人工智能的终极目标越来越近了。

面对市场上诸多的人工智能系统，以及背后的各种算法理论，使我想起了一部获奖的英国电影《模仿游戏》。这不是一部艺术上的 State of art，却赢得了第 87 届奥斯卡金像奖最佳改编剧本奖。在肯定这部作品的诸多因素中，我认为最重要的是，它宣誓了现阶段人工智能的本质：模仿。这也是本书自始至终贯穿的主题：模仿→象似性→算法理论。

但从另一个角度，我们希望能够终结一些问题，即便这些问题还未得到百分之百的解决（当然，从概率论的角度而言，没有百分之百），否则，我们很难进入以下阶段的研究，整个学科只会停滞不前。幸运的是，近些年，在序列标注上的全面突破，使我们有幸将目光放到了句子的范畴，最近提出的语义依存理论，更使汉语自然语言处理，无论理论还是实践都迎来了新的曙光。汉语的句子分析，终于跨越了句法的误区，走向了语义解析的道路。相信不久的将来，在语义解析的道路上，汉语 NLP 将会获得更大的突破。

本书的受众与特色

本书是一本研究汉语自然语言处理方面的基础性、综合性书籍，涉及 NLP 的语言理论、算法和工程实践的方方面面，内容繁杂。为此，我们设定本书的读者为如下几种：

- ❑ 具有一定计算机编程基础，对自然语言处理感兴趣的非专业人员。
- ❑ 希望构建完整的 NLP 应用系统的专业工程技术人员。
- ❑ 高校计算机专业和自然语言处理专业的大学生、研究生。
- ❑ 高校自然语言处理专业的教师。

需要指出的是，本书是一本系统介绍认知语言学和算法设计相结合的中文 NLP 书籍，并从认知语言学的视角重新认识和分析了 NLP 的句法和语义相结合的数据结构。这也是本书的创新之处。

内容及体系结构

为兼顾各方面的需求，我们对全书各部分做了精心的安排。从结构上，全书分为如下三大部分。

(1) 语言理论部分：涉及 4 个章节，第 2 章为汉语的发展历史；第 6 章为传统的句法理论；第 7 章为语料库和知识库的构建理论；第 8 章为认知语言学理论。

(2) 算法部分：涉及 4 个章节，第 3 章为中文分词算法；第 4 章为 NLP 中的概率图模型算法体系；第 6 章为句法的自动分析算法，包括转换生成语法的算法原理，以及依存句法的应用；第 9 章系统介绍了神经网络到深度学习算法体系，以及使用 LSTM 实现序列标注和依存句法。本书介绍的算法都提供开源的代码，具体下载地址已在每章介绍算法的时候指出，读者可参考书籍和网址的讲解内容进行调试，快速应用于实践中。

(3) 案例部分：涉及 4 个章节，第 1 章为开源 NLP 系统概览及入门代码；第 5 章为使用概率图模型算法进行词性标注、语义组块、命名实体识别等序列标注；第 9 章为使用 Word2Vec 的训练词向量模型；第 10 章为使用 SVM 进行长句切分、使用语义角色标注分析汉语句子等。

基本上每段理论讲解之后都辟出专门的案例讲解，以加深理论认识。对于重要的理论，甚至开辟专门的章节讲解其实现。案例分为两大部分，一部分是程序代码，读者可以参考书中的代码，将其直接应用到实践中；另一部分是语料，读者可以按书中指定的网络链接下载。

目 录

第 1 章 中文语言的机器处理.....	1
1.1 历史回顾.....	2
1.1.1 从科幻到现实	2
1.1.2 早期的探索	3
1.1.3 规则派还是统计派	3
1.1.4 从机器学习到认知 计算	5
1.2 现代自然语言系统简介	6
1.2.1 NLP 流程与开源框架	6
1.2.2 哈工大 NLP 平台及其 演示环境	9
1.2.3 Stanford NLP 团队及其 演示环境	11
1.2.4 NLTK 开发环境	13
1.3 整合中文分词模块	16
1.3.1 安装 Ltp Python 组件 ...	17
1.3.2 使用 Ltp 3.3 进行中文 分词	18
1.3.3 使用结巴分词模块	20
1.4 整合词性标注模块	22
1.4.1 Ltp 3.3 词性标注	23
1.4.2 安装 StanfordNLP 并 编写 Python 接口类	24

1.4.3 执行 Stanford 词性 标注	28
1.5 整合命名实体识别模块	29
1.5.1 Ltp 3.3 命名实体识别 ..	29
1.5.2 Stanford 命名实体 识别	30
1.6 整合句法解析模块	32
1.6.1 Ltp 3.3 句法依存树	33
1.6.2 Stanford Parser 类	35
1.6.3 Stanford 短语结构树 ...	36
1.6.4 Stanford 依存句法树 ...	37
1.7 整合语义角色标注模块	38
1.8 结语	40
第 2 章 汉语语言学研究回顾	42
2.1 文字符号的起源	42
2.1.1 从记事谈起	43
2.1.2 古文字的形成	47
2.2 六书及其他	48
2.2.1 象形	48
2.2.2 指事	50
2.2.3 会意	51
2.2.4 形声	53

2.2.5 转注	54	3.2.2 中文分词流程	99
2.2.6 假借	55	3.2.3 分词词典结构	103
2.3 字形的流变	56	3.2.4 命名实体的词典 结构	105
2.3.1 笔与墨的形成与变革 ...	56	3.2.5 词典的存储结构	108
2.3.2 隶变的方式	58	3.3 算法部分源码解析	111
2.3.3 汉字的符号化与结构 ...	61	3.3.1 系统配置	112
2.4 汉语的发展	67	3.3.2 Main 方法与例句	113
2.4.1 完整语义的基本 形式——句子	68	3.3.3 句子切分	113
2.4.2 语言的初始形态与 文言文	71	3.3.4 分词流程	117
2.4.3 白话文与复音词	73	3.3.5 一元词网	118
2.4.4 白话文与句法研究	78	3.3.6 二元词图	125
2.5 三个平面中的语义研究	80	3.3.7 NShort 算法原理	130
2.5.1 词汇与本体论	81	3.3.8 后处理规则集	136
2.5.2 格语法及其框架	84	3.3.9 命名实体识别	137
2.6 结语	86	3.3.10 细分阶段与最短 路径	140
第 3 章 词汇与分词技术	88	3.4 结语	142
3.1 中文分词	89	第 4 章 NLP 中的概率图模型	143
3.1.1 什么是词与分词规范 ...	90	4.1 概率论回顾	143
3.1.2 两种分词标准	93	4.1.1 多元概率论的几个 基本概念	144
3.1.3 歧义、机械分词、语言 模型	94	4.1.2 贝叶斯与朴素贝叶斯 算法	146
3.1.4 词汇的构成与未登录 词	97	4.1.3 文本分类	148
3.2 系统总体流程与词典结构	98	4.1.4 文本分类的实现	151
3.2.1 概述	98	4.2 信息熵	154

4.2.1 信息量与信息熵	154	4.6.1 随机场	193
4.2.2 互信息、联合熵、 条件熵	156	4.6.2 无向图的团 (Clique) 与因子分解	194
4.2.3 交叉熵和 KL 散度	158	4.6.3 线性链条件随机场	195
4.2.4 信息熵的 NLP 的 意义	159	4.6.4 CRF 的概率计算	198
4.3 NLP 与概率图模型	160	4.6.5 CRF 的参数学习	199
4.3.1 概率图模型的几个 基本问题	161	4.6.6 CRF 预测标签	200
4.3.2 产生式模型和判别式 模型	162	4.7 结语	201
4.3.3 统计语言模型与 NLP 算法设计	164	第 5 章 词性、语块与命名实体 识别	202
4.3.4 极大似然估计	167	5.1 汉语词性标注	203
4.4 隐马尔科夫模型简介	169	5.1.1 汉语的词性	203
4.4.1 马尔科夫链	169	5.1.2 宾州树库的词性标注 规范	205
4.4.2 隐马尔科夫模型	170	5.1.3 stanfordNLP 标注 词性	210
4.4.3 HMMs 的一个实例	171	5.1.4 训练模型文件	213
4.4.4 Viterbi 算法的实现	176	5.2 语义组块标注	219
4.5 最大熵模型	179	5.2.1 语义组块的种类	220
4.5.1 从词性标注谈起	179	5.2.2 细说 NP	221
4.5.2 特征和约束	181	5.2.3 细说 VP	223
4.5.3 最大熵原理	183	5.2.4 其他语义块	227
4.5.4 公式推导	185	5.2.5 语义块的抽取	229
4.5.5 对偶问题的极大似然 估计	186	5.2.6 CRF 的使用	232
4.5.6 GIS 实现	188	5.3 命名实体识别	240
4.6 条件随机场模型	193	5.3.1 命名实体	241
		5.3.2 分词架构与专名 词典	243

5.3.3 算法的策略——词典 与统计相结合	245	6.4 结语	310
5.3.4 算法的策略——层叠 式架构	252	第7章 建设语言资源库	311
5.4 结语	259	7.1 语料库概述	311
第6章 句法理论与自动分析	260	7.1.1 语料库的简史	312
6.1 转换生成语法	261	7.1.2 语言资源库的分类	314
6.1.1 乔姆斯基的语言观	261	7.1.3 语料库的设计实例: 国家语委语料库	315
6.1.2 短语结构文法	263	7.1.4 语料库的层次加工	321
6.1.3 汉语句类	269	7.2 语法语料库	323
6.1.4 谓词论元与空范畴	274	7.2.1 中文分词语料库	323
6.1.5 轻动词分析理论	279	7.2.2 中文分词的测评	326
6.1.6 NLTK 操作句法树	280	7.2.3 宾州大学 CTB 简介	327
6.2 依存句法理论	283	7.3 语义知识库	333
6.2.1 配价理论	283	7.3.1 知识库与 HowNet 简介	333
6.2.2 配价词典	285	7.3.2 发掘义原	334
6.2.3 依存理论概述	287	7.3.3 语义角色	336
6.2.4 Ltp 依存分析介绍	290	7.3.4 分类原则与事件 分类	344
6.2.5 Stanford 依存转换、 解析	293	7.3.5 实体分类	347
6.3 PCFG 短语结构句法分析	298	7.3.6 属性与分类	352
6.3.1 PCFG 短语结构	298	7.3.7 相似度计算与实例	353
6.3.2 内向算法和外向 算法	301	7.4 语义网与百科知识库	360
6.3.3 Viterbi 算法	303	7.4.1 语义网理论介绍	360
6.3.4 参数估计	304	7.4.2 维基百科知识库	364
6.3.5 Stanford 的 PCFG 算法 训练	305	7.4.3 DBpedia 抽取原理	365
		7.5 结语	368

第 8 章 语义与认知	370	8.5.3 构式知识库	417
8.1 回顾现代语义学	371	8.6 结语	420
8.1.1 语义三角论	371	第 9 章 NLP 中的深度学习	422
8.1.2 语义场论	373	9.1 神经网络回顾	422
8.1.3 基于逻辑的语义学	376	9.1.1 神经网络框架	423
8.2 认知语言学概述	377	9.1.2 梯度下降法推导	425
8.2.1 象似性原理	379	9.1.3 梯度下降法的实现	427
8.2.2 顺序象似性	380	9.1.4 BP 神经网络介绍和 推导	430
8.2.3 距离象似性	380	9.2 Word2Vec 简介	433
8.2.4 重叠象似性	381	9.2.1 词向量及其表达	434
8.3 意象图式的构成	383	9.2.2 Word2Vec 的算法 原理	436
8.3.1 主观性与焦点	383	9.2.3 训练词向量	439
8.3.2 范畴化: 概念的 认知	385	9.2.4 大规模上下位关系的 自动识别	443
8.3.3 主体与背景	390	9.3 NLP 与 RNN	448
8.3.4 意象图式	392	9.3.1 Simple-RNN	449
8.3.5 社交中的图式	396	9.3.2 LSTM 原理	454
8.3.6 完形: 压缩与省略	398	9.3.3 LSTM 的 Python 实现	460
8.4 隐喻与转喻	401	9.4 深度学习框架与应用	467
8.4.1 隐喻的结构	402	9.4.1 Keras 框架介绍	467
8.4.2 隐喻的认知本质	403	9.4.2 Keras 序列标注	471
8.4.3 隐喻计算的系统 架构	405	9.4.3 依存句法的算法 原理	478
8.4.4 隐喻计算的实现	408	9.4.4 Stanford 依存解析的 训练过程	483
8.5 构式语法	412		
8.5.1 构式的概念	413		
8.5.2 句法与构式	415		

9.5 结语	488
第 10 章 语义计算的架构	490
10.1 句子的语义和语法预处理	490
10.1.1 长句切分和融合	491
10.1.2 共指消解	496
10.2 语义角色	502
10.2.1 谓词论元与语义角色	502
10.2.2 PropBank 简介	505
10.2.3 CPB 中的特殊句式	506
10.2.4 名词性谓词的语义角色	509
10.2.5 PropBank 展开	512
10.3 句子的语义解析	517
10.3.1 语义依存	517
10.3.2 完整架构	524
10.3.3 实体关系抽取	527
10.4 结语	531

1.1.2 早期的探索

第 1 章

中文语言的机器处理

自然语言处理（Natural Language Processing, NLP）是研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法，也是人工智能领域中一个最重要、最艰难的方向。说其重要，因为它的理论与实践与探索人类自身的思维、认知、意识等精神机制密切相关；说其艰难，因为每一项大的突破都历经十年乃至几十年以上，要耗费几代人的心血。

近些年，NLP 在中文分词、词性标注、词汇语义、句法解析方面均获得了很大的突破。大量的技术都应用于商业实践，并在商业领域获得了良好的市场和经济效益。文本方面主要有：基于自然语言理解的智能搜索引擎和智能检索、智能机器翻译、自动摘要与文本综合、文本分类与文件整理、自动阅卷系统、信息过滤与垃圾邮件处理、文学研究与古文研究、语法校对、文本数据挖掘与智能决策、基于自然语言的计算机程序设计等。语音方面主要有：机器同声传译、智能客户服务、聊天机器人、语音挖掘与多媒体挖掘、多媒体信息提取与文本转化、对残疾人智能帮助系统等。

这使得从事研究这个行业的科技人员越来越多，本书的目的之一就是为使这些朋友尽快进入这个领域而降低门槛。为此，笔者不想在开篇谈那些艰深的理论知识和数学原理，这些沉重的主题还是留在后面为读者慢慢道来。我们先简要回顾一下 NLP 的历史，然后通过几个开源系统，领略一下当代 NLP 的风采。

1.1 历史回顾

世界上早期的科幻小说作者，英国诗人雪莱的夫人玛丽·雪莱，于1818年创作了科幻小说《弗兰肯斯坦——现代普罗米修斯的故事》。小说讲的是，一个叫作弗兰肯斯坦的年轻科学家希望利用所学的生物学知识，制造出一个类人生物。在强烈的科学探索欲望的驱使下，他从停尸房等处取得不同人体的器官和组织，拼合成一个人体，并利用雷电的电激活了这个人造的生命。经过电击，人造人瞬间被弗兰肯斯坦赋予了生命。刚刚诞生的人造人天性善良并向往美好。不过，由于相貌丑陋，社会并不接纳它，并将其视作怪物。因为他与社会的种种矛盾，终于导致怪物走上了报复之路。故事的结尾是悲惨的，弗兰肯斯坦因病而死，怪物也自焚消失。

1.1.1 从科幻到现实

这里想要探讨的并不是故事在文学上的价值，而是其在科学上的重要意义。自工业革命以来，在生产生活的各个领域，科技都走入了人们的视野：人们创造了铁路系统、公路系统以延伸人的双腿；创造了各种各样的机械臂装置来模仿人的双手；创造了五颜六色的纺织品模仿动物的皮毛；还发展了电影和广播系统来愉悦视听。两百年来，这部作品给我们留下一个发人深省的启示：似乎我们所有的科学技术活动都围绕着一个方向——人类如何创造而发展自身！

时间过得很快，这个问题第二次摆在我们面前的时候，从《弗兰肯斯坦》发表又过了一百多年。经过二次大战的洗礼，人类对自身的认识进入了一个新的阶段。在民主制度与专制制度、民族主义与殖民主义的斗争中，科学技术得到了空前的发展。1950年，计算理论的先驱阿兰·麦席森·图灵写了一篇著名的论文——《计算机器与智能》，其内容是：如果电脑能在5分钟内回答由人类测试者提出的一系列问题，且其超过30%的回答让测试者误认为是人类所答，则电脑通过测试（来自百度百科）。这就是著名的“图灵测试”。计算机专业的读者对于图灵测试应该并不陌生。这个测试想从实验的角度提出一个假设：“机器能与人类交流吗？”问题听起来似乎有点悬，我们换一种方式来重新描述一下这个问题：“有可能设计出具有类似人类智能的机器吗？”

虽然从现代视野来看，图灵问题本身显得有些粗糙，但不得不承认，图灵问题的提出是人类科技的一个重要的里程碑。它揭开了科学幻想那遥不可及的面纱，将人工智能最重要的任务赤裸裸地摆在了人们的面前——所谓人工智能的终极任务就是人类要制造出具有人类语言和思考能力的机器。

1.1.2 早期的探索

计算机刚一诞生,人们就开始着手研究用它来解析人类的自然语言。这一需求不仅源于科学家的个人兴趣,而且具有重要的战略意义:20 世纪 50 年代开始,大家都意识到以美、苏两国为首的两大政治集团迟早要进入冷战时代。此时,美国就尝试着利用计算机将大量俄语资料自动翻译成英语,以窥探苏联科技的最新发展。虽然当时的计算机还在襁褓之中,但研究者从破译军事密码中得到启示,简单地认为语言之间的差异只不过是对“同一语义”的不同编码而已,从而想当然地采用译码技术解析不同的语言。这就是最早机器翻译理论的思想。

1954 年 1 月 7 日,美国乔治敦大学和 IBM 公司首先成功地将 60 多句俄语自动翻译成英语。当时的系统还非常简单,仅包含 6 个语法规则和 250 个词。但是,由于媒体的广泛报道,美国政府备受鼓舞,认为这是一个巨大的进步,长期发展将具有重要的战略意义。而实验者声称:

在三到五年之内就能够完全解决从一种语言到另一种语言的自动翻译问题。

当时普遍认为只要制定好各种翻译规则,通过大量规则的堆砌就能完美地实现语言间的自动翻译。1956 年,美国语言学家 N. Chomsky 从 Shannon 的工作中利用了有限状态马尔科夫过程的思想,首先把有限状态自动机作为一种工具来刻画语言的语法,并且把有限状态语言定义为由有限状态语法生成的语言。这些早期的研究工作产生了“形式语言理论”(Formal Language Theory)。它为最初的机器翻译工作提供了理论基础。

经过近十年的努力,机器翻译并未获得本质性的突破。1964 年美国科学院成立了语言自动处理咨询委员会(ALPAC),开始了为期两年的综合调查分析和测试。直到 1966 年年底,委员会公布了一个题为《语言与机器》的报告(简称 ALPAC 报告)。该报告全面否定了机器翻译的可行性,并建议停止对机器翻译项目的资金支持。这一报告的发表终结了自然语言处理的第一个时代——机器翻译时代。

1.1.3 规则派还是统计派

虽然机器翻译时代结束了,但自然语言处理这一新兴学科(NLP)却没有消亡。时间进入 20 世纪七八十年代后,随着经济发展特别是国际市场机制的成熟,国与国之间的语言障碍越来越成为更深层次国际交流的壁垒。传统的人工作业方式已经不能满足需求,这就需要一种自动机器来取代人工。同时,计算机硬件技术大幅度提高,使中等规模的语料(百万级)处理成为可能。经过十多年的发展,自然语言处理逐渐作为人工智能的

一个独立领域而发展起来，此时的自然语言处理也分为两种不同的派别。

一种是以语言学理论为基础，根据语言学家对语言现象的认识，采用规则形式描述或解释歧义行为或歧义特性，称为规则派。规则派的方法通常是基于乔姆斯基的语言理论的。它通过语言所必须遵守的一系列原则来描述语言，以此来判断一个句子是正确的（遵循语言原则）还是错误的（违反语言原则）。规则派首先要对大量的语言现象进行研究，归纳出一系列的语法规则。然后再形成一套复杂的规则集——语言分析或生成系统，对自然语言进行分析处理。

另一种是以基于语料库的统计分析为基础的经验主义方法，也称为统计派，该方法更注重用数学，从能代表自然语言规律的大规模真实文本中发现知识，抽取语言现象或统计规律。统计派来源于多种数学基础，包括通香农（Shannon）的信息论、最优化方法、概率图模型、神经网络、深度学习等。它将语言事件赋予概率，作为其可信度，由此来判断某个语言现象是常见的还是罕见的。统计派的方法则偏重于对语料库中人们实际使用的普通语言现象的统计表述。统计方法是语料库语言学研究的主要内容。

两派曾经一度相执不下。这里不考虑两派之间孰是孰非，而是希望通过一个著名的实验给大家一点启示，这个实验就是著名的约翰·塞尔的中文屋子实验。

一个对中文一窍不通的、以英语为母语的人被关闭在一间只有两个通口的封闭屋子中。屋子里有一本用英文写成、从形式上说明中文文字句法和文法组合规则的手册及一大堆中文符号。屋子外的人不断向屋子内递进用中文写成的问题。屋子内的人便按照手册的说明，将中文符号组合成对问题的解答，并将答案递出屋子。

约翰·塞尔认为，尽管屋子里的人甚至可以做到以假乱真，让屋子外的人以为他是中文的母语用户，然而，他压根就不懂中文。而在上述过程中，屋子外的人所扮演的角色相当于程序员，屋子中的人相当于计算机，而那本手册则相当于计算机程序。

正如屋子中的人不可能通过手册理解中文一样，计算机也不可能通过程序来获得对自然语言（中文）的理解能力。塞尔由此得出结论：图灵测试中机器根本不理解回答的问题，机器根本没有思考，机器也没有智能。（来自网络文摘）

塞尔的中文屋测试本来是针对图灵测试的一个反驳意见，但它所揭示的意义是深刻的。当时所谓的人工智能，特别是对自然语言处理领域的主要任务，不过是使用机器来解析人类的语言符号，将其转换为机器能够处理的形式和结构，在机器内部按照人们已经设定好的逻辑进行处理，最后将处理的结果再转码为人类理解的形式，传输给人类。这与大多数非智能的计算机程序没有本质的不同。

的确, 计算机几十年的发展, 绝大多数程序不都是这样吗? 即便像操作系统这样高度复杂的软件, 也不能说其中的哪段代码能够自主地识别设备、完成请求任务; 或者为任务的执行提出合理性或哪怕看起来稍微有点自发的智能行为。所谓智能不过是程序人员对程序执行的某种预先的设定, 所有看起来智能的行为都是在确定性条件下的一条执行路径。

难道, 研究就停留在这里了吗? 想要突破这一点确实是很艰难的。但是, 科学家的脚步并没有就此停止。之后人们终于把视野从确定性的问题开始转向随机性问题, 实践上从单纯的指令系统转向研究人类大脑的机制——认知科学。古人云: 知人者智, 自知者明。在科学探索的艰辛道路上, 智慧是一种较低层次的能力, 而自省才具有更高级的境界。

1.1.4 从机器学习到认知计算

进入 20 世纪 90 年代, 世界经济从国际化逐渐走向了一体化的进程, 随着互联网的普遍应用, 国际社会之间的交流越来越频繁。人们足不出户即可通过互联网了解世界上发生的大事及形形色色的生活。这种交流不仅出现在政治、经济上, 还渗透到人们的日常生活之中。中文互联网搜索引擎的基础语言构件就是中文分词。没有大规模、高精度的分词, 很难想象能够实现精准的中文搜索。NLP 技术不再是实验室中易碎的花瓶, 而是登堂入室地走进了千家万户。

伴随着这些突破的是一系列新方法(算法体系)的出现, 它们被统称为“机器学习”。这些方法大多都以神经元和大脑的工作原理为理论基础, 模拟人类的认知行为而发展起来。人们发现这些程序不是编程编出来的而是训练出来的, 经过二三十年, 机器学习在处理多维、非线性问题方面取得了精确而稳定的效果。例如, 在大规模语料上的中文分词、词性标注问题的解决, 使中文信息检索和文本挖掘成为可能。因为大多数的机器学习方法都以统计学为基础, 毫无疑问, 统计派占了上风。

时间进入 21 世纪, 终于在 2006 年, 以 Hinton 为首的几位科学家历经近 20 年的努力, 终于成功设计了第一个多层神经网络算法, 因其通过多层架构实现了抽象认知的学习能力, Hinton 将其命名为“深度学习”。

深度学习就是一种特征学习方法, 把原始数据通过一些简单的但是是非线性的模型转变成更高层次的、更加抽象的表达。通过足够多的转换的组合, 非常复杂的函数也可以被学习。(选自《深度学习-LeCun、Bengio 和 Hinton 的联合综述》)

在多年的实验中，人们发现了认知的两个重要机制：一个是抽象，另一个是迭代。从原始信号，做低层抽象，逐渐向高层抽象迭代，在迭代中抽象出更高层的模式。这是认知的生物学原理。目前来看，深度学习在解决机器视觉和语音识别方面都获得了非常好的效果，相关的技术都已经商业化。所以，人们评价，通过深度学习理论及算法，人类终于找到了如何处理“抽象概念”这个亘古难题的方法。

作为认知计算的重要起点，深度学习的递归神经网络在自然语言处理方面同样获得了成功。虽然在中文领域离商业化还有距离，但是这个距离应该不会太远。

1.2 现代自然语言系统简介

多年来，人们通过对大脑的研究发现，人类大脑控制语言的功能区有两个：一个是位于前脑的布罗卡区，另一个是位于后脑的威尔尼克区。布罗卡区主管语言的处理、话语的生成；威尔尼克区的主要功能是分辨语音、形成语义，它与语言的解析有密切的关系。而有关信息的搜索、推理和决策则由大脑的前额叶完成。直观上，一个自然语言处理系统也应包含最少三个模块：语言的解析、语义的理解及语言的生成，如图 1.1 所示。

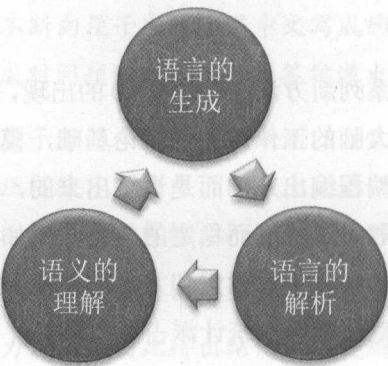


图 1.1 自然语言系统的整体逻辑

1.2.1 NLP 流程与开源框架

近些年来，随着互联网的发展，计算机处理自然语言的需求变得愈来愈迫切，除比较悠久的机器翻译之外，自然语言处理在信息检索、信息抽取、数据挖掘、舆情分析、文本摘要、自动问答系统等方面都获得了很广泛的应用。自然语言系统在实际研究中，

涉及的领域要广泛得多，系统结构也变得越来越复杂，但其根本都是围绕着这三个方面而展开的。

总体来讲，现代自然语言处理的主要任务已经跨越对词的研究，发展到了对句子研究，即句法、句义及句子生成的研究，已经能够比较好地解决句子层面的问题，但还未达到完全解决篇章层面的问题。目前，计算机处理自然语言系统尚不足以达到较为自由地进行人机交互的程度。

因此，有关自然语言，特别是语义方面的诸多问题仍未解决。在过去几十年的研究中，已经产生了许多专业技术，每一项都作用于语言理解的不同层面和不同任务。例如，这些技术包括完全句法分析、浅层句法分析、信息抽取、词义消歧、潜在语义分析、文本蕴含和指代消解。这些技术都不能完美或完全地译解出语言的本义。与程序语言不同，人类语言不具备严整的逻辑结构。由于语言的高度语境化和隐喻化的性质，人类自身对于特定语言表达的本义往往也会有不同的理解。

人们对于文本的理解，很大程度上受到主题、作者及时间等相关背景知识的影响——篇章的范畴。想一想这句话：“咬死了猎人的狗。”这是 NLP 中一个经典的歧义范例。究竟是“[[咬死了猎人]的狗]”还是“[咬死了][猎人的狗]”呢？如果不依赖于足够的语境知识，人们将很难给出结论。

再如，“人生的旅程”、“生命的终点”、“花一般的姑娘”、“花钱如流水”等都使用了隐喻手法，它们的共性是，句子的意义是不能从字面的意义直接得到的。

在语法解析层面，大规模高精度的中文分词、词性标注系统已经基本达到商用要求，但在句法解析方面还存在精度问题。

在语义解析层面，命名实体识别、语义块都已经获得了较高的精度。人工智能对知识库的研究历史悠久，已经形成一整套的知识库的架构和推理体系。实现句子到知识库的主要方法是语义角色标注系统，但在整句的理解层面，语义角色标注系统的精度严重依赖于句法解析系统，这使该系统离商用还有一段距离。

由于前两个层面的问题，语言生成的发展相对滞后，应用也不广泛，虽有商业化的应用，但是范围都非常狭窄，基本都集中在机器翻译领域。即使近些年，也未出现较有影响力的汉语语言生成的商业系统。因此，关于语言生成不作为本书主要讨论的内容。

然而，自然语言处理发展的短短几十年来，仍旧取得了很大的发展，一些系统成功地用于搜索引擎、文本挖掘（文本分类、文本聚类等）、舆情分析、推荐系统等领域。本

节主要针对这些较为成熟的自然语言模块和系统提出了一个粗略的系统架构，并给出一些优秀的开源系统供大家研究学习。

图 1.2 所示为自然语言处理的一般架构。该图中，左侧是语法层面的模块，包括中文分词、词性标注及句法解析。大家对这部分应该比较熟悉。

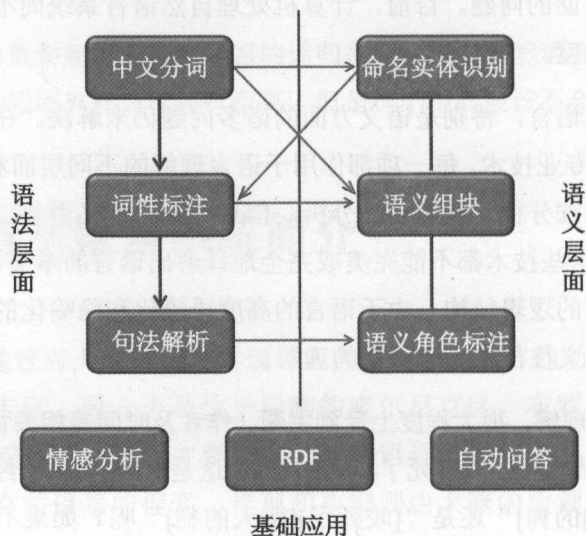


图 1.2 自然语言处理的一般架构

右侧则偏重于语义层面，命名实体识别主要用来识别语料中专有名词和未登录词的成词情况，如人名、地名、组织机构名称等，也包括一些特别的专名。该图中来自左侧的箭头表示命名实体识别受到中文分词和词性标注的影响。换句话说，准确的命名实体识别是以准确的分词和词性标注为前提的。

语义组块用来确定一个以上的词汇构成的短语结构，即短语级别的标注，主要识别名词性短语、动词性短语、介词短语等，以及其他类型的短语结构。本书的范畴也涉及小句结构。语义组块的自动识别来源于中文分词、词性标注和命名实体识别的共同信息，也就是说，语义组块的识别特征必须包含中文分词（命名实体识别）和词性标注两部分。

语义角色标注是以句子中的谓语动词为中心预测出句子中各个语法成分的语义特征，是句子解析的最后一个环节，也是句子级别语义研究的重要里程碑。语义组块、语义角色标注等分析结果，可以通过机器学习方法转换为知识库中的 RDF 形式，并直接用于自动问答系统。上述框架中，语义角色标注直接受到句法解析和语义组块的影响。从中文分词阶段到语义角色标注阶段大约经历了 4~5 个依次串连的模块。这导致语义角色标注

的精度显著降低。基于上述架构的语义角色标注系统尚达不到商业应用的范畴。

为了使读者有一个直观的认识，在详细讲解 NLP 的各模块之前，首先简要介绍比较流行的几个开源的中文 NLP 系统。近些年，发布出来的 NLP 开源系统很多，其中，NLP 全系列处理框架如表 1.1 所示。

表 1.1 NLP全系列处理框架

名 称	包含模块和下载地址	开发语言
哈工大的Ltp3.X	中文分词、词性标注、未登录词识别、句法分析、语义角色标注 网址: https://github.com/HIT-SCIR/ltp/releases	C++
Stanford NLP	中文分词、词性标注、未登录词识别、句法分析等 网址: http://nlp.stanford.edu/software/index.shtml	Java
FudanNLP	中文分词、句法分析等 网址: https://code.google.com/p/fudannlp/	Java
HanLP	中文分词、句法分析等各类算法 网址: https://github.com/hankcs/HanLP	Java
.....

优秀的中文分词框架，如 NLP 分词框架，如表 1.2 所示。

表 1.2 NLP分词框架

名 称	包含模块和下载地址	开发语言
ICTCLAS分词系统	具有里程碑意义的中文分词系统 网址: http://www.threedweb.cn/forum-2-1.html	C++
Ansj中文分词系统	中等规模的中文分词系统 网址: https://github.com/NLPchina/ansj_seg	Java
结巴分词	小规模中文分词 网址: https://github.com/fxsjy/jieba	Python
.....

上述系统除提供源码之外，很多都提供网络的访问接口，有的是 API；有的是 Web 页面；有的两者兼而有之。这里介绍两大著名的 NLP 系统的网络平台。

1.2.2 哈工大 NLP 平台及其演示环境

哈工大语言技术平台（Language Technology Platform，LTP）是哈工大社会计算与信息检索研究中心开发的一整套中文语言处理系统。语言技术平台提供包括中文分词、词

性标注、命名实体识别、依存句法分析、语义角色标注等丰富、高效、精准的自然语言处理技术。经过哈工大社会计算与信息检索研究中心 11 年的持续研发和推广，LTP 已经成为国内外最具影响力的中文处理基础平台，曾获 CoNLL 2009 七国语言句法语义分析评测总成绩第一名，中文信息学会钱伟长一等奖等重要成绩和荣誉。目前，LTP 已经被 500 多家国内外研究机构和企业使用，多家大企业和科研机构付费使用。

2011 年 6 月 1 日，为了与业界同行共同研究和开发中文信息处理核心技术，该中心正式将 LTP 开源。

哈工大语言云演示平台（见图 1.3）：<http://www.ltp-cloud.com/demo/>，又称为哈工大“语言云”，是以哈工大社会计算与信息检索研究中心研发的“语言技术平台（LTP）”为基础，为用户提供高效、精准的中文自然语言处理云服务。使用“语言云”非常简单，只需根据 API 参数构造 HTTP 请求即可在线获得分析结果，而无须下载 SDK、无须购买高性能的机器，同时支持跨平台、跨语言编程等。2014 年 11 月，哈工大联合科大讯飞公司共同推出“哈工大—讯飞语言云”，为广大用户提供电信级稳定性和支持全国范围网络接入的语言云服务，有效支持包括中小企业在内的开发者的商业应用需要。

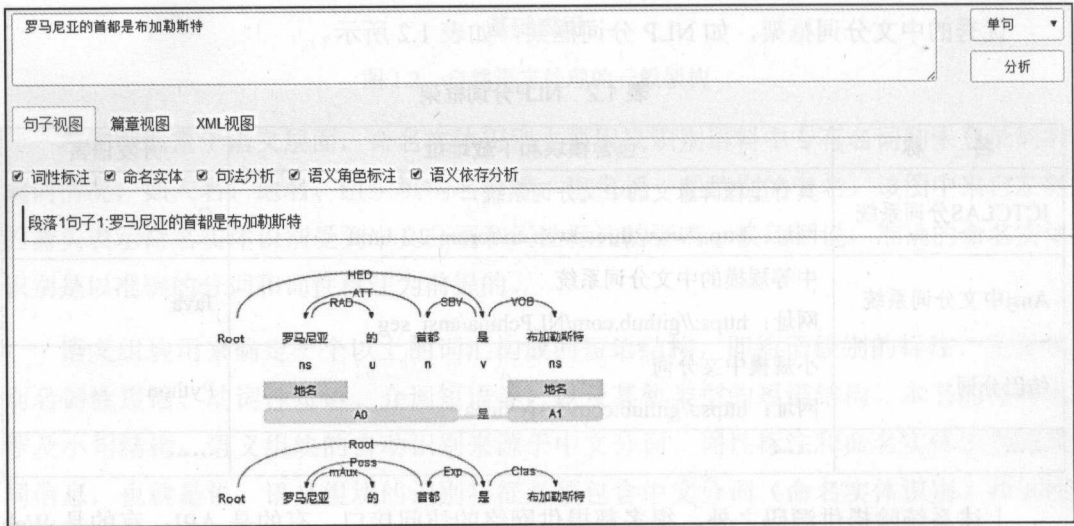


图 1.3 哈工大语言云演示平台

该语言平台不仅能够提供基于中文分词、词性标注、命名实体识别、句法分析、语义角色标注的全套自然语言处理框架，而且可以通过可视化的图形输出，使用户一目了然。

1.2.3 Stanford NLP 团队及其演示环境

斯坦福 (Stanford) 自然语言处理团队 (网址: <http://nlp.stanford.edu/>) 是一个由斯坦福大学的教师、科研人员、博士后、程序员组成的团队。该团队致力于研究计算机理解人类语言的工作。从计算语言学的基础研究到人类语言的关键技术都是其工作范围, 涵盖诸如句子的理解、机器翻译、概率解析和标注、生物医学信息抽取、语法归纳、词义消歧、自动问答及文本区域到 3D 场景的生成等。

斯坦福大学的 NLP 团队包括语言学系和计算机科学系, 以及人工智能实验室的成员。斯坦福自然语言处理团队的一个显著特点, 是能够有效地处理复杂而深刻的语言模型和数据分析, 创造性地将 NLP 与概率和机器学习算法进行组合。其研究成果主要集中于多语种、跨语言的广泛而稳定的技术。这些技术包括指代消解系统、词性标注系统、高性能的概率句法解析器、生物命名实体识别系统和关于阿拉伯文、中文、德文的文本算法。

该团队提供的软件分发包均用 Java 编写而成。2014 年 10 月至今的版本需要运行在 Java8+ 上 (2013 年 3 月至 2014 年 9 月的版本需要 Java1.6+; 2005 年至 2013 年 2 月的版本需要 Java1.5+)。分发程序包包括一个命令行调用 (.bat 和 .sh)、jar 文件、Java API 文档及源代码。任何用户都可从 <http://nlp.stanford.edu/software/index.shtml> 处下载相关代码。一些相关团队, 如 NLTK 扩展了 Stanford NLP 团队的工作, 将其与 Python 进行绑定。

Stanford NLP 开源项目涉及广泛的 NLP 应用, 并在某些中文 NLP 应用中具有卓越的性能, 一些主要的中文 NLP 应用如下。

1) 斯坦福句法解析器

概率自然语言句法解析器包括 PCFG (与概率的上下文无关的短语) 和依存句法解析器, 一个词汇的 PCFG 解析器, 以及一个超快速的神经网络的依存句法解析器和深度学习重排序器。斯坦福还提供了一个在线句法分析器演示: <http://nlp.stanford.edu:8080/parser/>, 以及神经网络的依存解析器文档和常见问题解答。

2) 斯坦福命名实体识别器

该识别器基于条件随机场序列模型, 用于英文、中文、德文、西班牙文的连同命名实体识别, 以及一个在线 NER 演示。

3) 斯坦福词性标注器

基于最大熵 (CMM) 算法, 词性标注 (POS) 系统包括英语、阿拉伯语、汉语、法语、德语和西班牙语。

4) 斯坦福分词器

斯坦福分词器是一个基于 CRF 算法的分词器，支持阿拉伯语和汉语。

除此之外，斯坦福还提供包括机器翻译、分类器和 GUI 等其他的一些应用。前文已经介绍过一些同样优秀的分词系统，从本节开始，简要介绍一下斯坦福的其他语言处理模块，以及与 Python NLTK 平台的整合。

同样，著名的斯坦福 NLP 团队也提供了一个全系列语言平台，网址为 <http://corenlp.run/>。目前该平台仅支持英文。我们找到另一个支持中文的句法解析平台，Stanford NLP Parser 演示平台：<http://nlp.stanford.edu:8080/parser/index.jsp>，执行结果如图 1.4 所示。

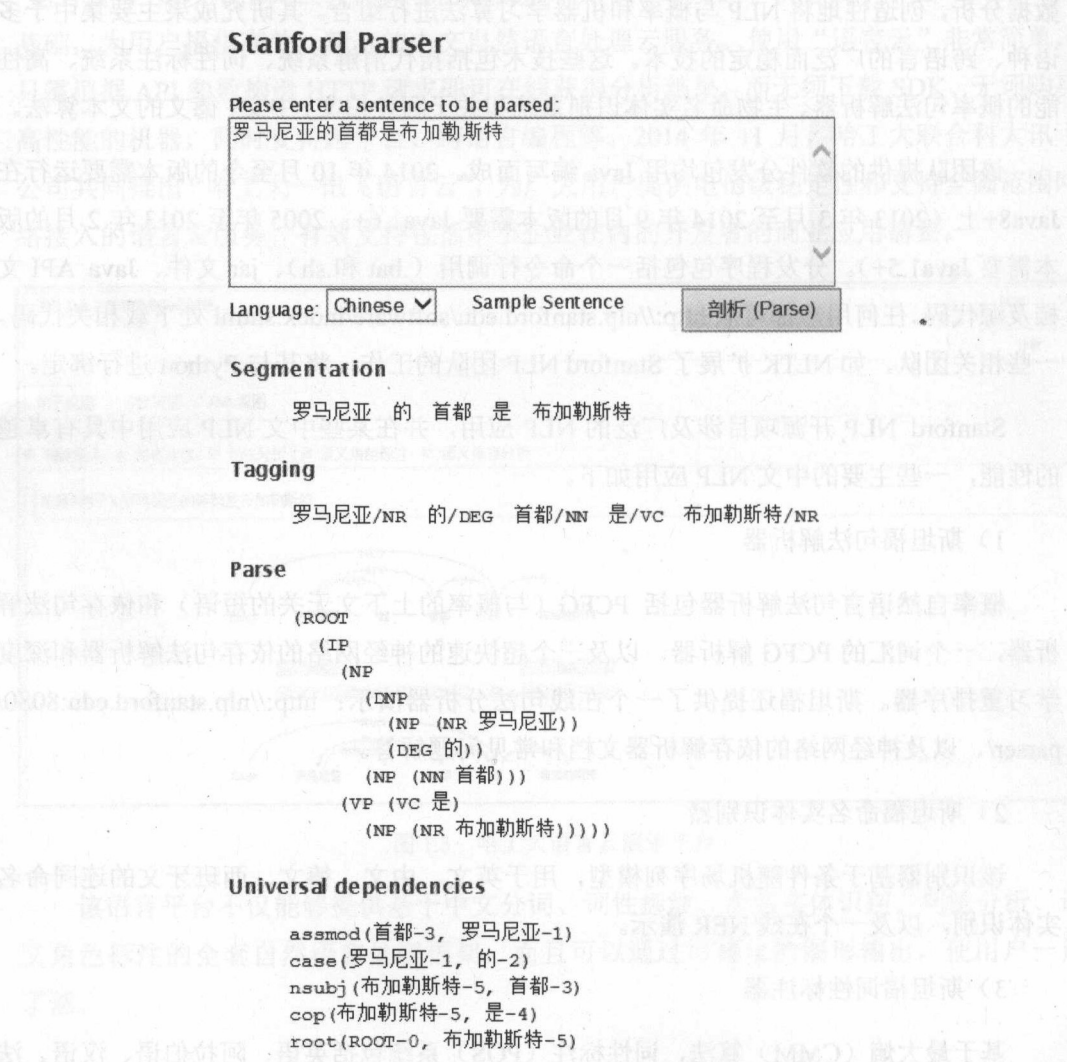


图 1.4 Stanford 句法解析演示平台

该平台提供包括中文分词、词性标注、多种句法解析结果的综合语言处理平台。

1.2.4 NLTK 开发环境

除这些比较好的开源应用系统之外，还需要建立一个统一的 NLP 的开发平台。作为一个整合平台，我们希望该平台具有如下特征。

- ❑ 能够跨不同操作系统，即平台可至少运行于 Windows 和 Linux 发行版的操作系统中。对这两个主流的操作系统完全兼容。如果不加说明，本书中的代码均可运行于这两个操作系统中。
- ❑ 可屏蔽各种计算机语言的差异性，即可以自由地整合基于 Java、C++、Python 语言等的大多数 NLP 系统，而无须将开源系统重构为某种单一的计算机语言。
- ❑ 该平台需要预先包含绝大多数的 NLP 常用功能，即各类基于规则的或基于统计的词法、句法、语义等分析模块，有助于我们方便地调试各类语言系统，并得出相应的结果，以减少代码量。
- ❑ 由于语言模型资源会占用较大内存，该平台应具备较小的系统内存开销。

这类开源平台有几家，考虑到资源占用的经济性，比较突出的是 NLTK 系统。NLTK 是使用 Python 程序构建的自然语言数据工作的主要平台。它提供了易于使用的界面，以超过 50 种语料库和词汇资源，如 Penn TreeBank（宾夕法尼亚大学树库一部分）、Penn PropBank（宾夕法尼亚大学谓词论元库一部分）、WordNet 等，还有一套分类、分词、词干、标注、分析和语义推理的基本程序框架，并包装为工业强度的 NLP 库文本处理库，以及一个活跃的论坛。NLTK 全面涉及计算语言学的各类技术，适合语言专家、工程师、学生、教育工作者、研究人员和行业用户使用，并受到一致好评。NLTK 可用于 Windows、Mac OS X 和 Linux。最重要的是，NLTK 是一个免费、开源的社区驱动的项目。读者除了下载无须支付其他任何费用。

为全面讲解平台的使用，NLTK 组织出版了有关语言库使用的书籍《Python 自然语言处理》。该书的中文版已经发行，在各大书籍网站均有销售。受篇幅限制，本书较少提及语言处理的入门知识，有关入门知识可参考此书。

下面简要讲解如何安装 NLTK 系统。

1) 操作系统

本教程使用的操作系统是 Windows 10 64 位（如果你使用的是 32 位操作系统，则应升级为 64 位的操作系统；如果你安装的是 Windows 7 或 XP，则无须升级到 Windows 10

操作系统) 或 Linux CentOS 64 位。

2) 安装 Python 开发环境

读者可从 <https://www.python.org/downloads/> 下载相应的版本。目前, 最新版本是 python-2.7.11, 如果是在 Windows 下则执行 python-2.7.11.msi 文件进行安装。

注: 除标准安装之外, 还有 Anaconda 跨平台安装包, 读者可从 <https://www.continuum.io/downloads> 下载, 因为 Anaconda 已经默认安装了 NLTK, 所以不需要再安装专门的 NLTK 安装包。

3) 安装常用的 Python 应用程序

Python 开发环境建立完成之后, 一般可在命令行下直接使用 `pip install ×××package` 语句下载常用的安装程序。这里向读者推荐一个 python 常用包下载网址, 从这里读者可以找到一些常用的, 但 pip 库中没有的 python 包, 读者可以从 <http://www.lfd.uci.edu/~gohlke/pythonlibs/> 网址上下载对应的编译好的安装包。

(1) 安装数学运算包。

① python 环境。

```
pip install numpy
```

② anaconda 环境。

```
conda install scipy
```

这条指令会安装全部的 mkl、numpy、scipy、scikit-learn 等高性能数学计算软件包。

(2) 安装 MySQL 数据库工具包 (<http://www.lfd.uci.edu/~gohlke/pythonlibs/>): 下载后使用 pip + 包路径安装。

MySQL-python, a Python database API 2.0 interface for the MySQL database

MySQLclient is a Python 3 compatible fork of MySQL-python.

[MySQL python-1.2.5-cp27-none-win32.whl](#)

[MySQL python-1.2.5-cp27-none-win amd64.whl](#)

MySQLclient, a fork of the MySQL-python interface for the MySQL database.

[mysqlclient-1.3.7-cp27-none-win32.whl](#)

[mysqlclient-1.3.7-cp27-none-win amd64.whl](#)

[mysqlclient-1.3.7-cp34-none-win32.whl](#)

[mysqlclient-1.3.7-cp34-none-win amd64.whl](#)

[mysqlclient-1.3.7-cp35-cp35m-win32.whl](#)

[mysqlclient-1.3.7-cp35-cp35m-win amd64.whl](#)

(3) 安装 Tornado 网络包。

```
pip install Tornado
```

除上述必要的安装包之外，读者也可根据自身的要求选择其他常用的软件包下载安装。

4) 安装 NLTK 开发环境

(1) NLTK 安装语言开发系统的基本安装指令。

```
pip install nltk
```

(2) 下载常用的一些语料库：NLTK 安装之后，执行如下代码可下载相关的语料。

```
import sys
import os
import nltk

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

nltk.download()
```

系统会出现如图 1.5 和图 1.6 所示的界面：这个平台默认使用的都是英文语料库。

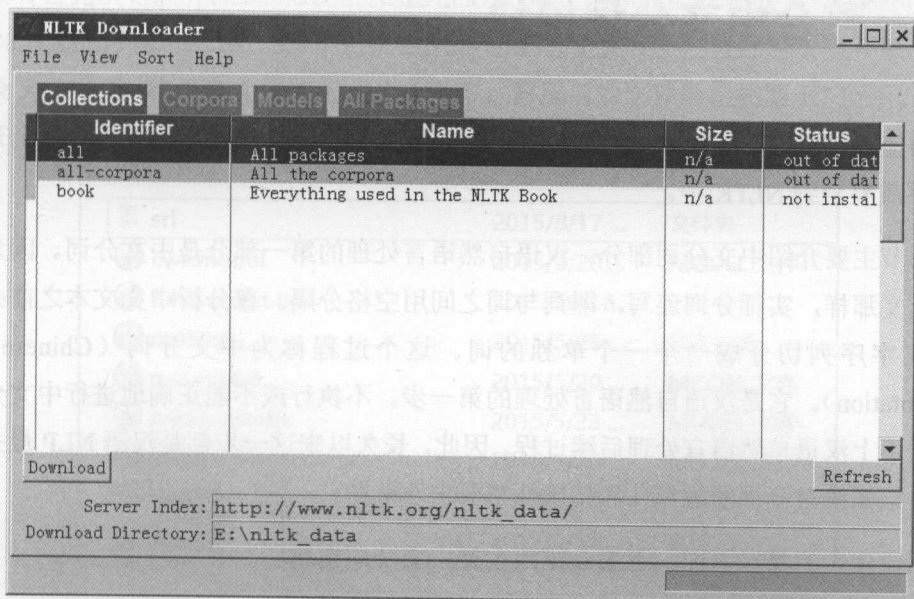


图 1.5 NLTK Download 语料库 1

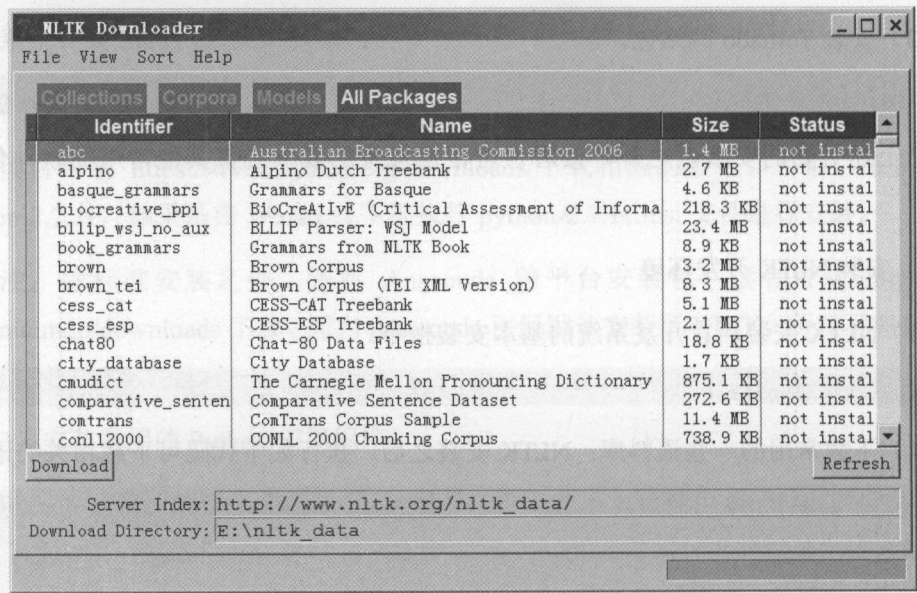


图 1.6 NLTK Download 语料库 2

读者可根据出现的界面选择指定的语料或语料库，全部语料库下载解压后大小共 2GB 多。因此，就中文自然语言处理环境而言，没有必要安装所有的语料库，需要什么就安装什么。可根据研究方向的不同安装不同的资源。

1.3 整合中文分词模块

为使读者深入了解 NLP 的各个功能模块，从本节开始，举例介绍如何将几个开源的 NLP 系统整合到 NLTK 中。

本节主要介绍中文分词部分。汉语自然语言处理的第一部分是中文分词。因为中文不像英文那样，实行分词连写，即词与词之间用空格分隔。在分析中文文本之前必须将一个汉字序列切分成一个一个单独的词。这个过程称为中文分词（Chinese Word Segmentation）。它是汉语自然语言处理的第一步。不执行或不能正确地进行中文分词，根本谈不上汉语自然语言处理后续过程。因此，长久以来这一步都是汉语 NLP 最关键的环节。该系统对分词的规模性和精确性都要求非常高。

按照使用的算法不同，下面介绍两大类中文分词模块。

- ❑ 基于条件随机场（CRF）的中文分词算法的开源系统。
- ❑ 基于张华平 NShort 的中文分词算法的开源系统。

1.3.1 安装 Ltp Python 组件

国内使用 CRF 做中文分词的开源系统主要为哈工大的 HIT LTP 语言技术平台。该项目的源代码可从 <https://github.com/HIT-SCIR/ltp/releases/tag/v3.3.0> 获取，源代码用 C++编写，对源代码感兴趣的读者可下载用于学习。本书的后面章节会对 LTP 的源代码做进一步解析，读者可以先下载源码备用。

对应于该源代码的语言模型库可从 <http://pan.baidu.com/share/link?shareid=1988562907&uk=2738088569#dir/path=%252Fltp-models> 下载。目前最高版本为 3.3 版。

源代码和语言模型包括：中文分词、词性标注、未登录词识别、依存句法、语义角色标注几个模块。

这里要将项目与 Python 整合，HIT LTP 也提供了该项目与 Python 扩展包。扩展包的源码和案例可从 <https://github.com/HIT-SCIR/pyltp> 下载。一般我们都直接使用已经预编译的 pyltp 安装包，可参考：<https://pypi.python.org/pypi/pyltp> 网页。

(1) pyltp 安装。

```
pip install pyltp
```

(2) 部署语言模型库。

读者首先从 <http://pan.baidu.com/share/link?shareid=1988562907&uk=2738088569#dir/path=%252Fltp-models%252F3.3.0> 下载 Ltp 3.3 版的模型文件。在自己的 X 盘下新建一个目录 X:\ltp3.3，将模型文件解压到此目录下。解压后的文件结构如图 1.7 所示。

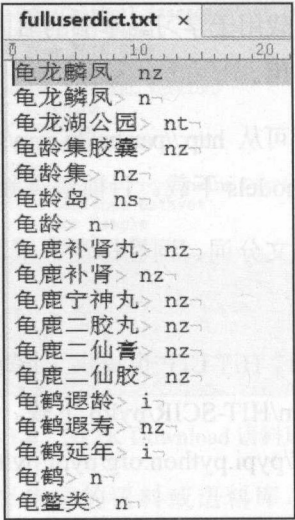
名称	修改日期	类型
sr	2015/8/17 ...	文件夹
cws.model	2015/5/20 ...	MODEL 文件
fulluserdict.txt	2015/10/18 ...	TXT 文件
md5.txt	2015/5/23 ...	TXT 文件
ner.model	2015/5/20 ...	MODEL 文件
parser.model	2015/5/23 ...	MODEL 文件
pos.model	2015/5/20 ...	MODEL 文件
userdict.txt	2015/8/20 ...	TXT 文件
version	2015/5/19 ...	文件

图 1.7 解压后的文件结构

如图 1.7 所示，cws.model 为中文分词模块所需的语言模型，fulluserdict.txt 为用户可

添加的外部词典文件。cws.model 是一个二进制文件，有关文件的内容，将在后面章节详细讲解。

fulluserdict.txt 是用户可添加的外部词典文件，图 1.8 给出该文件的词汇结构。



fulluserdict.txt x	
龟龙麟凤	nz
龟龙麟凤	n
龟龙湖公园	nt
龟龄集胶囊	nz
龟龄集	nz
龟龄岛	ns
龟龄	n
龟鹿补肾丸	nz
龟鹿补肾	nz
龟鹿宁神丸	nz
龟鹿二胶丸	nz
龟鹿二仙膏	nz
龟鹿二仙胶	nz
龟鹤遐龄	i
龟鹤遐寿	nz
龟鹤延年	i
龟鹤	n
龟鳖类	n

图 1.8 Ltp 用户词典样例

第一列是词，第二列到第 n 列是该词的候选词性，如果仅用于分词，该词的词性可以忽略。由于 Ltp 3.3 能够支持大约 50 万个常用词的中文分词，并提供高于 95% 以上的准确结果，因此，如果仅用于实验和中小规模的文本语料，该分词器足以满足需求。

Ltp 的用户词典可放置一些不能正确切分的专用词汇，但该用户词典不属于外部词典，有专门的算法给予支持，并与 CRF 算法相融合。有关更多的细节使用，可参照：http://ltp.readthedocs.org/zh_CN/latest/ltp-test.html。

1.3.2 使用 Ltp 3.3 进行中文分词

(1) Ltp 3.3 安装成功之后，新建一个 Python 文件如下。

```
# -*- coding: utf-8 -*-
import sys
import os
from pyltp import Segmentor # 导入 ltp 库

reload(sys)
sys.setdefaultencoding('utf-8') # 设置 UTF-8 输出环境
```



```

model_path = "X:\\ltp3.3\\cws.model" #Ltp3.3 分词模型库
segmentor = Segmentor() #实例化分词模块
segmentor.load(model_path) # 加载分词库

words = segmentor.segment("在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。")

print " | ".join(words) # 分割后的分词结果

```

这里不使用外部词典，仅用 Ltp 3.3 进行中文分词。

分词结果如下。

```

在 | 包含 | 问题 | 的 | 所有 | 解 | 的 | 解 | 空间 | 树 | 中 | ， | 按照 | 深度 | 优先 | 搜索 | 的 | 策略 | ， | 从 | 根节点 | 出发 | 深度 | 探索 | 解 | 空间 | 树 | 。

```

(2) 分词结果的后处理。

观察上述分词结果，“解 | 空间”、“解 | 空间 | 树”、“深度 | 优先”都可以看作一个完整的专有名词：解空间、解空间树、深度优先，而分词器划分的粒度过细。为了获得更精确的结果可以将错分的结果合并为专有名词。这就是分词结果的后处理过程，即一般外部用户词典的构成原理。

```

..... # 这部分与上面的代码相同：导入 ltp 库；设置 UTF-8 输出环境

postdict={"解 | 空间":"解空间","深度 | 优先":"深度优先"}
# 分词后处理--矫正一些错误的结果
model_path = "X:\\ltp3.3\\cws.model" #Ltp3.3 分词库
segmentor = Segmentor()
segmentor.load(model_path)
words = segmentor.segment("在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。")
seg_sent = " | ".join(words)
for key in postdict:
    seg_sent=seg_sent.replace(key,postdict[key])
print seg_sent

```

分词结果如下。

```

在 | 包含 | 问题 | 的 | 所有 | 解 | 的 | 解空间 | 树 | 中 | ， | 按照 | 深度优先 | 搜索 | 的 | 策略 | ， | 从 | 根节点 | 出发 | 深度 | 探索 | 解空间 | 树 | 。

```

(3) 现在加入用户词典，词典中登录一些新词，如解空间。

```

..... # 这部分与上面的代码相同：导入 ltp 库；设置 UTF-8 输出环境

```

```
model_path = "X:\\ltp3.3\\cws.model"
user_dict = "X:\\ltp3.3\\fulluserdict.txt" #外部专有名词词典路径
segmentor = Segmentor()
segmentor.load_with_lexicon(model_path,user_dict) #加载专有名词词典
sent = "在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。"
words = segmentor.segment(sent)
print " | ".join(words)
```

其分词结果如下。

```
在 | 包含 | 问题 | 的 | 所有解 | 的 | 解空间 | 树 | 中 | ， | 按照 | 深度优先 | 搜索 | 的 |
策略 | ， | 从 | 根节点 | 出发 | 深度 | 探索 | 解空间 | 树 | 。
```

这里使用的用户词典不是基于词匹配的外部词典，而是内置的专有名词词典。有关专有名词词典的编写，需要根据应用场景的不同来制作。CRF 类分词器的专有名词识别被称为领域自适应问题，这部分的算法将在后面章节给出详细讲解。

1.3.3 使用结巴分词模块

张华平 NShort 的中文分词算法是目前大规模中文分词的主流算法。在商用领域，大多数搜索引擎公司都使用该算法作为主要的分词算法。具有算法原理简单、容易理解、便于训练、大规模分词的效率高、模型支持增量扩展、模型占用资源低等优势。

这里使用的结巴分词器是该算法的 Python 实现，结巴分词的算法核心就是 Nshort 中文分词算法。

结巴分词可从 <https://github.com/fxsjy/jieba> 目录下载。结巴分词模块可支持如下三种分词方式。

- ☐ 精确模式，试图将句子最精确地切开，适合文本分析（类似 Ltp 的分词方式）。
- ☐ 全模式，把句子中所有可以成词的词语都扫描出来，速度非常快，但是不能解决歧义。
- ☐ 搜索引擎模式，在精确模式的基础上对长词再次切分，提高召回率，适合用于搜索引擎分词。
- ☐ 支持繁体分词。
- ☐ 支持基于概率的用户词典。

(1) 结巴分词库的安装。

```
.pip install jieba
```

(2) 使用结巴分词。

```

# -*- coding: utf-8 -*-
import sys
import os
import jieba # 导入结巴分词库

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

# 结巴分词--全模式
sent = '在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。'
wordlist = jieba.cut(sent, cut_all=True)
print " | ".join(wordlist)

# 结巴分词--精确切分
wordlist = jieba.cut(sent) # cut_all=False
print " | ".join(wordlist)

# 结巴分词--搜索引擎模式
wordlist = jieba.cut_for_search(sent)
print " | ".join(wordlist)

```

分词结果如下。

```

Building prefix dict from ...
Dumping model to file cache ...
Loading model cost 3.178 seconds.
Prefix dict has been built succesfully.
在 | 包含 | 问题 | 的 | 所有 | 解 | 的 | 解空 | 空间 | 树 | 中 | | | 按照 | 深度 | 优先
| 搜索 | 的 | 策略 | | | 从 | 根结 | 节点 | 点出 | 出发 | 深度 | 探索 | 索解 | 解空 | 空间 |
树 | |

在 | 包含 | 问题 | 的 | 所有 | 解 | 的 | 解 | 空间 | 树中 | , | 按照 | 深度 | 优先 | 搜索
| 的 | 策略 | , | | 从根 | 节点 | 出发 | 深度 | 探索 | 解 | 空间 | 树 | 。

在 | 包含 | 问题 | 的 | 所有 | 解 | 的 | 解 | 空间 | 树中 | , | 按照 | 深度 | 优先 | 搜索
| 的 | 策略 | , | | 从根 | 节点 | 出发 | 深度 | 探索 | 解 | 空间 | 树 | 。

```

结巴分词的基础词库较之 Ltp 分词要少一些，标准词典的词汇量约有 35 万个。上例的分词结果显示，一些专有名词的切分缺乏足够的精度，不仅出现粒度问题，还出现错分问题（如上例中的“树中”、“从根”）。由此可见，分词器的精度不仅受到算法的影响，更受到语言模型库的规模影响。这一点对于 NShort 最短路径算法尤其明显。

(3) 定义用户词典。

如图 1.9 所示，词典格式为一个词占一行；每一行分为三部分：词语、词频和词性，用空格隔开，顺序不可颠倒。Userdict.txt 文件必须为 UTF-8 编码。

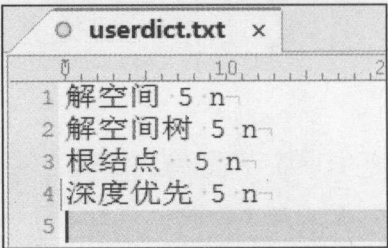


图 1.9 结巴分词用户词典样例

结巴分词中的词典词频设置，默认为 5。这个数越大，说明该字符串成词的概率越高，受到内置词典的干扰就越小；这个数越小，用户词典内的词受到内置词典的干扰越强，不能正确切分的概率就越大。用户需要根据实际情况来设置这个值。

(4) 使用用户词典。

```
..... # 这部分与上面的代码相同：导入 ltp 库；设置 UTF-8 输出环境

jieba.load_userdict("userdict.txt") # 加载外部 用户词典
sent = '在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。'
wordlist = jieba.cut(sent) # 结巴分词--精确切分
print " | ".join(wordlist)
```

分词结果如下。

```
Building prefix dict from ...
Dumping model to file cache ...
Loading model cost 3.178 seconds.
Prefix dict has been built succesfully.
在 | 包含 | 问题 | 的 | 所有 | 解 | 的 | 解空间树 | 中 | ， | 按照 | 深度优先 | 搜索 | 的 |
策略 | ， | 从 | 根节点 | 出发 | 深度 | 探索 | 解空间树 | 。
```

1.4 整合词性标注模块

词性标注 (Part-of-Speech Tagging 或 POS Tagging)，又称为词类标注，是指判断出在一个句子中每个词所扮演的语法角色。例如，表示人、事物、地点或抽象概念的名称

就是名词；表示动作或状态变化的词为动词；用来描写或修饰名词性成分或表示概念的性质、状态、特征或属性的词称为形容词，等等。

在汉语中，常用词的词性都不是固定的，也就是说，一个词可能具有多个词性，并且对于每个不同的词性，汉语词汇都没有形态的变化，这就为确定中文词性带来困难；但在另一方面，从整体上看，大多数词语，特别是实词，一般只有一两个词性，而且位于第一位词性的频次远远高于第二位的词性，即使选取最高词频作为唯一词频，系统也可实现 80% 准确率。因此，中文词性标注中影响词性标注精度的因素主要是要正确判断文本中那些常用词的词性。

本节主要介绍几个中文词性标注系统的应用，一般而言，中文的词性标注算法比较统一，大多数使用 HMM（隐马尔科夫模型）或最大熵算法，如前文中的结巴分词的词性标注实现。为了获得更高的精度，也有使用 CRF 算法的，如 Ltp 3.3 中的词性标注。虽然分词与词性标注是两个不同的模块，但是在一般的工程应用中，语料的中文分词和词性标注通常同时完成。为了讲解的连续性，本节仍使用 Ltp 3.3 中的词性标注模块。

目前流行的中文词性标签有两大类：北大词性标注集和宾州词性标注集（后面章节将具体讲解两类标注集的规范和转换方法）。两类标注方式各有千秋，为了更全面地反映中文标注的概况，我们使用 Stanford 大学的中文词性标注模块作为另一个中文词性标注系统。

1.4.1 Ltp 3.3 词性标注

继续使用 Ltp 3.3 中文词性标注模块。图 1.7 列出的语言包列表中，词性标注模块的文件名为 pos.model。这里使用 1.3 节的分词结果进行标注。

```
# -*- coding: utf-8 -*-
import sys
import os
from pyltp import *
# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')
```

```
sent = "在 包含 问题 的 所有 解 的 解空间树 中 ， 按照 深度优先 搜索 的 策略 ， 从 根节点 出
发 深度 探索 解空间树 。"
```

```
words = sent.split(" ")
```

```
postagger = Postagger() # 实例化词性标注类
postagger.load("X:\\ltp3.3\\pos.model") # 导入词性标注模型
postags = postagger.postag(words)
for word,postag in zip(words,postags):
    print word+"/"+postag,
```

词性标注结果如下。

```
在/p 包含/v 问题/n 的/u 所有/b 解/v 的/u 解空间树/n 中/nd , /wp 按照/p 深度优先/d 搜索/v
的/u 策略/n , /wp 从/p 根节点/n 出发/v 深度/n 探索/v 解空间树/v 。 /wp
```

标注的标签与原词用“/”分隔，每个标签都有其语法的意义。例如，“n”表示名词，“v”表示动词。Ltp 的词性标注遵从北大词性标注规范。有关词性标注的标签意义后面将有专门的章节进行讲解。

1.4.2 安装 StanfordNLP 并编写 Python 接口类

下面介绍如何使用 StanfordNLP 系统来进行中文词性标注。在使用之前，需要安装 jdk 环境，Stanford-CoreNLP-3.6.0 版本需要安装 Java 8+的环境。有关源程序可从 Oracle 的 JDK 网站中下载：<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>。

如图 1.10 所示，方框内为必须下载的 JDK Win x64 位系统。因为篇幅所限，这里省略了 JDK 的安装和环境变量的设置。如果读者对此安装不很清楚，建议百度一下：配置 JAVA 环境变量。

Java SE Development Kit 8u65		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.		
Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	jdk-8u65-linux-arm32-vfp-hflt.tar.gz
Linux ARM v8 Hard Float ABI	74.66 MB	jdk-8u65-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.67 MB	jdk-8u65-linux-i586.rpm
Linux x86	174.84 MB	jdk-8u65-linux-i586.tar.gz
Linux x64	152.69 MB	jdk-8u65-linux-x64.rpm
Linux x64	172.86 MB	jdk-8u65-linux-x64.tar.gz
Mac OS X x64	227.14 MB	jdk-8u65-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.71 MB	jdk-8u65-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.01 MB	jdk-8u65-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.22 MB	jdk-8u65-solaris-x64.tar.Z
Solaris x64	96.74 MB	jdk-8u65-solaris-x64.tar.gz
Windows x86	181.24 MB	jdk-8u65-windows-i586.exe
Windows x64	186.57 MB	jdk-8u65-windows-x64.exe

图 1.10 下载 JDK 环境

部署完成 JDK 开发环境，接下来安装 Stanford NLP 的语言程序包。读者可从 <http://stanfordnlp.github.io/CoreNLP/> 网页下载全套语言处理模块。本书使用的是 Stanford CoreNLP 3.6.0 版。Stanford 全套语言包下载链接如图 1.11 所示。

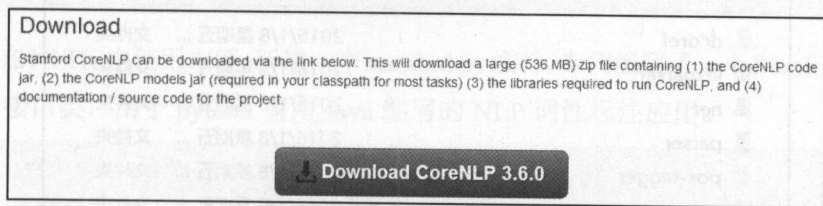


图 1.11 Stanford 全套语言包下载链接

下载文件名为 stanford-corenlp-full-2015-12-09.zip，但 stanford-corenlp-full-2015-12-09.zip 只携带了英文的语言模型包，中文部分的语言模型需要单独下载，下载网址为 <http://nlp.stanford.edu/software/CRF-NER.shtml>。Stanford 全套中文模型包下载链接如图 1.12 所示。

We also provide Chinese models built from the Ontonotes Chinese named entity data. There are two models, one using distributional similarity clusters and one without. These are designed to be run on word-segmented Chinese. So, if you want to use these on normal Chinese text, you will first need to run Stanford Word Segmenter or some other Chinese word segmenter, and then run NER on the output of that!

Chinese models

图 1.12 Stanford 全套中文模型包下载链接

在 X 盘下建立一个名为 stanford-corenlp 的目录，将下载的 stanford-corenlp-full-2015-12-09.zip 解压到此目录中。可以看到如图 1.13 所示的文件夹。

名称	修改日期	类型
input.txt	2015/12/10 星期...	TXT 文件
LICENSE.txt	2015/12/10 星期...	TXT 文件
README.txt	2015/12/10 星期...	TXT 文件
ejml-0.23.jar	2015/12/10 星期...	WinRAR 压缩文件
javax.json.jar	2015/12/10 星期...	WinRAR 压缩文件
javax.json-api-1.0-sources.jar	2015/12/10 星期...	WinRAR 压缩文件
joda-time.jar	2015/12/10 星期...	WinRAR 压缩文件
joda-time-2.9-sources.jar	2015/12/10 星期...	WinRAR 压缩文件
jollyday.jar	2015/12/10 星期...	WinRAR 压缩文件
jollyday-0.4.7-sources.jar	2015/12/10 星期...	WinRAR 压缩文件
protobuf.jar	2015/12/10 星期...	WinRAR 压缩文件
slf4j-api.jar	2015/12/10 星期...	WinRAR 压缩文件
slf4j-simple.jar	2015/12/10 星期...	WinRAR 压缩文件
stanford-corenlp-3.6.0.jar	2015/12/10 星期...	WinRAR 压缩文件
stanford-corenlp-3.6.0-javadoc.jar	2015/12/10 星期...	WinRAR 压缩文件
stanford-corenlp-3.6.0-models.jar	2015/12/10 星期...	WinRAR 压缩文件
stanford-corenlp-3.6.0-sources.jar	2015/12/10 星期...	WinRAR 压缩文件
xom.jar	2015/12/10 星期...	WinRAR 压缩文件
xom-1.2.10-src.jar	2015/12/10 星期...	WinRAR 压缩文件
build.xml	2015/12/10 星期...	XML 文档
input.txt.xml	2015/12/10 星期...	XML 文档
pom.xml	2015/12/10 星期...	XML 文档

图 1.13 Stanford-CoreNLP 目录（部分）

其中，stanford-corenlp.jar 为主执行文件。下载的中文模型文件 stanford-chinese-corenlp-2015-12-08-models.jar 解压后的目录结构如图 1.14 所示。

名称	修改日期	类型
dcoref	2016/1/8 星期五 ...	文件夹
lexparser	2016/1/8 星期五 ...	文件夹
ner	2016/1/8 星期五 ...	文件夹
parser	2016/1/8 星期五 ...	文件夹
pos-tagger	2016/1/8 星期五 ...	文件夹
segmenter	2016/1/8 星期五 ...	文件夹
srparser	2016/1/8 星期五 ...	文件夹

图 1.14 下载的中文模型文件 stanford-chinese-corenlp-2015-12-08-models.jar 解压后的目录结构

stanford-chinese-corenlp-2015-12-08-models.jar 中的中文模型全部解压到 X:\stanford-corenlp 的 models 目录中。其中，pos-tagger 目录下放置了词性标注的中文模型。

为了演示方便，本书默认的操作系统为 Windows 10，读者可以从 <http://nlp.stanford.edu/software/tagger.shtml> 处下载 stanford-postagger.zip 包，并找到 stanford-postagger.bat 作为执行命令行的参考脚本。

```
java -mx300m -cp "stanford-postagger.jar;" edu.stanford.nlp.tagger.maxent.MaxentTagger
-model %1 -textFile %2
```

该命令行脚本可分为如下几部分。

```
java -mx300m # 调用 java 程序，以及设置的最大内存
-cp "stanford-postagger.jar;" # 调用的 jar 包，3.6 版共有三个 jar 包，分别为
stanford-postagger.jar、slf4j-api.jar、slf4j-simple.jar
edu.stanford.nlp.tagger.maxent.MaxentTagger # 最大熵分类器
-model %1 # 最大熵模型文件
-textFile %2 # 最大熵输入文本文件
```

在命令行下，重新编辑 bat 文件，并使用 1.3 节分词的结果，代码如下。

```
java -mx300m -cp "X:\\ stanfordpostagger\\stanford-postagger.jar; X:\\
stanfordpostagger\\lib\\slf4j-api.jar; X:\\ stanfordpostagger\\lib\\slf4j-simple.
jar" edu.stanford.nlp.tagger.maxent.MaxentTagger -model "X:\\ stanfordpostagger\\
models\\chinese-distsim.tagger" -textFile posttest.txt > result.txt
```

其中，posttest.txt 文件内容如下。

在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。

输出的结果文件 result.txt 如下。

在#P 包含#VV 问题#NN 的#DEC 所有#DT 解#VV 的#DEC 解空间树#NN 中#LC , #PU 按照#P 深度优先#NN 搜索#NN 的#DEC 策略#NN , #PU 从#P 根节点#NN 出发#VV 深度#JJ 探索#NN 解空间树#VV 。#PU

为了在 NLTK 中使用方便,新建一个 stanford.py 文件,专门编写了一个 StanfordCoreNLP 的若干个接口类,用于 python 调用 Java 编写的 NLP 词性标注应用。

```
# -*- coding: utf-8 -*-
import sys
import os

# CoreNLP 3.6 jar 包和中文模型包
# ejml-0.23.jar、javax.json.jar、jollyday.jar、joda-time.jar、jollyday.jar、
# protobuf.jar、slf4j-api.jar
# slf4j-simple.jar、stanford-corenlp-3.6.0.jar、xom.jar
class StanfordCoreNLP(): # 所有 StanfordNLP 的父类
    def __init__(self, jarpath):
        self.root = jarpath
        self.tempsrcpath = "tempSrc" # 输入临时文件路径
        self.jarlist = ["ejml-0.23.jar", "javax.json.jar", "jollyday.jar", "joda-
time.jar", "protobuf.jar", "slf4j-api.jar", "slf4j-simple.jar", "stanford-corenlp-3.6
.0.jar", "xom.jar"]
        self.jarpath = ""
        self.buildjars()

    def buildjars(self): # 根据 root 路径构建所有的 jar 包路径
        for jar in self.jarlist:
            self.jarpath += self.root+jar+";"

    def savefile(self, path, sent): # 创建临时文件存储路径
        fp = open(path, "wb")
        fp.write(sent)
        fp.close()

    def delfile(self, path): # 删除临时文件
        os.remove(path)

class StanfordPOSTagger(StanfordCoreNLP): # 词性标注子类
    def __init__(self, jarpath, modelpath):
        StanfordCoreNLP.__init__(self, jarpath)
        self.modelpath = modelpath # 模型文件路径
```



```

        self.classfier = "edu.stanford.nlp.tagger.maxent.MaxentTagger" # 词性标注主类

        self.delimiter = "/" # 标签分隔符
        self.__buildcmd()

    def __buildcmd(self): # 构建命令行
        self.cmdline = 'java -mxlg -cp "'+self.jarpath+'" '+self.classfier+'
-model "'+self.modelpath+'" -tagSeparator '+self.delimiter

    def tag(self,sent): #标注句子
        self.savefile(self.tempsrcpath,sent)
        tagtxt = os.popen(self.cmdline+" -textFile '"+self.tempsrcpath,'r').
read() # 结果输出到变量中
        self.delfile(self.tempsrcpath)
        return tagtxt

    def tagfile(self,inputpath,outputpath):# 标注文件
        os.system(self.cmdline+' -textFile '+inputpath+' > '+outputpath )

```

1.4.3 执行 Stanford 词性标注

使用 Stanford POSTagger 类来进行词性标注。

```

# -*- coding: utf-8 -*-
import sys
import os
from stanford import StanfordPOSTagger

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

root = 'E:/nltk_data/stanford-corenlp/'
modelpath = root+"models/pos-tagger/chinese-distsim/chinese-distsim.tagger"
st = StanfordPOSTagger(root,modelpath)
seg_sent = '在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根节点出发深度探索解空间树。'
taglist = st.tag(seg_sent)
print taglist

```

输出结果如下。

在/P 包含/VV 问题/NN 的/DEC 所有/DT 解/VV 的/DEC 解空间/NN 树/NN 中/LC，/PU 按照/P 深

度优先/NN 搜索/NN 的/DEC 策略/NN, /PU 从/P 根节点/NN 出发/VV 深度/JJ 探索/NN 解空间/NN 树/NN。/PU

在 Stanford POSTagger 的分词结果中,“P”表示介词;“NN”表示一般名词;“LC”表示方位词等,由于 Ltp 3.3 和 Stanford 使用的词性标签不同,这里对标注结果不做评估。在后面章节中会对词性标注集做一个专门的评估。

1.5 整合命名实体识别模块

命名实体识别用于识别文本中具有特定意义的实体,常见的实体主要包括人名、地名、机构名及其他专有名词等。本书将命名实体识别划分在语义范畴的原因是,命名实体识别不仅需要标注词的语法信息(名词),更重要的是要指示词的语义信息(人名还是组织机构名等)。这里所需要识别的命名实体一般不是指已知名词(词典中的登录词),而是指新词(或称未登录词)。

文本中新词的涌现反映了人类词汇的能产性。所谓能产性指基本词(字)能够构成其他新词,但是这些新词的产生并非没有规律性。应该讲,不同的义类具有不同的规律:构成中国人名的统计规律,显然区别于外文译名,或其他专有名词。新词构成的概率分布按照其义类的不同而有所不同。这是命名实体构成的普遍规律。

更具体的命名实体识别任务还要识别出文本中三大类(实体类、时间类和数字类)、七小类(人名、机构名、地名、时间、日期、货币和百分比)命名实体。

本章继续使用 1.4 节介绍的两个系统——Ltp 3.3 和 StanfordNLP 中的命名实体识别模块,但在例句上做了一些变化,使用了含有专名—地名的句子。

1.5.1 Ltp 3.3 命名实体识别

继续使用 Ltp 3.3 中文命名实体识别模块。图 1.7 列出的语言包列表中,命名实体识别模块的文件名为 ner.model。这里使用新的已分词例句。

执行程序如下。

```
# -*- coding: utf-8 -*-
import sys
import os
```

```

from pyltp import *

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

sent = "欧洲 东部 的 罗马尼亚 ， 首都 是 布加勒斯特 ， 也 是 一 座 世界性 的 城市 。"
words = sent.split(" ")
postagger = Postagger()
postagger.load("X:\\ltp3.3\\pos.model") # 导入词性标注模块
postags = postagger.postag(words)

recognizer = NamedEntityRecognizer()
recognizer.load("X:\\ltp3.3\\ner.model") # 导入命名实体识别模块
netags = recognizer.recognize(words, postags)

for word, postag, netag in zip(words, postags, netags): # 输出结果
    print word+"/"+postag+"/"+netag,

```

识别结果如下。

```

欧洲/ns/S-Ns 东部/nd/O 的/u/O 罗马尼亚/ns/S-Ns , /wp/O 首都/n/O 是/v/O 布加勒斯特
/ns/S-Ns , /wp/O 也/d/O 是/v/O 一/m/O 座/q/O 世界性/n/O 的/u/O 城市/n/O 。 /wp/O
[INFO] 2016-01-06 19:01:47 build-config: add 61 constrains.
[INFO] 2016-01-06 19:01:47 report: number of labels 13

```

输出例句中“/”为分隔符，分隔符将结果按词为单位分为三段。例如，第一段是词“欧洲”，第二段是词性“ns”，第三段“S-Ns”就是识别的专名。这里标签“O”表示非专名，“S-Ns”表示地名。全部标签在后面的章节中有详细说明。

1.5.2 Stanford 命名实体识别

如果仅使用斯坦福的中文命名实体识别模块（也称为 NER），可从 <http://nlp.stanford.edu/software/CRF-NER.shtml> 下载。与词性标注的不同之处在于，命名实体识别模块的程序和中文模型库分开存放。应用程序可以从图 1.15 所示的截图处下载。

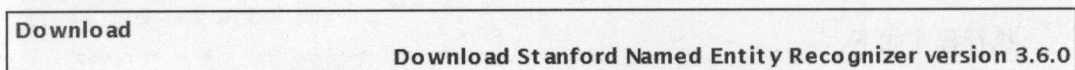


图 1.15 截图

斯坦福命名实体识别也可以通过 Stanford-CoreNLP 包运行。中文模型库内使用了基

于词向量的语义相似度模型。运行该模型的前提条件需要首先进行中文分词，然后将分词结果（以空格分开的分词文本）作为输入，再运行命名实体识别模块进行输出。

在图 1.14 目录列表中的名为 ner 的子目录中，放置着 stanford-chinese-corenlp-2015-12-08-models.jar 解压出来的 chinese.misc.distsim.crf.ser.gz 基于 crf 的命名实体识别中文语言模型。

最后，仿照词性标注模块，编写一个 Python 命名实体类如下。

```
class StanfordNERTagger(StanfordCoreNLP):
    def __init__(self, modelpath, jarpath):
        StanfordCoreNLP.__init__(self, jarpath)
        self.modelpath = modelpath # 模型文件路径
        self.classfier = "edu.stanford.nlp.ie.crf.CRFClassifier"
        self.__buildcmd()
    # 构建命令行
    def __buildcmd(self):
        self.cmdline = 'java -mxlg -cp "'+self.jarpath+'" '+self.classfier+'
-loadClassifier "'+self.modelpath+'"'
    #标注句子
    def tag(self, sent):
        self.savefile(self.tempsrcpath, sent)
        tagtxt = os.popen(self.cmdline+' -textFile '+self.tempsrcpath, 'r').
read() # 输出到变量中
        self.delfile(self.tempsrcpath)
        return tagtxt
    # 标注文件
    def tagfile(self, sent, outpath):
        self.savefile(self.tempsrcpath, sent)
        os.system(self.cmdline+' -textFile '+self.tempsrcpath+' > '+outpath)
        self.delfile(self.tempsrcpath)
```

执行代码如下。

```
# -*- coding: utf-8 -*-
import sys
import os
from stanford import StanfordNERTagger
reload(sys)
sys.setdefaultencoding('utf-8') # 设置 UTF-8 输出环境

root = 'E:/nltk_data/stanford-corenlp/'
modelpath = root+'models/ner/chinese.misc.distsim.crf.ser.gz'
```

```
st = StanfordNERTagger(modelpath, root)
seg_sent = '欧洲 东部 的 罗马尼亚 , 首都 是 布加勒斯特 , 也 是 一 座 世界性 的 城市 。 '
taglist = st.tagfile(seg_sent, "ner_test.txt")
print taglist
```

输出结果为 ner_test.txt 文件, 打开该文件可以看到输出结果如下。

```
欧洲/LOC 东部/O 的/O 罗马尼亚/GPE , /O 首都/O 是/O 布加勒斯特/GPE , /O 也/O 是/O 一/O 座
/O 世界性/O 的/O 城市/O 。 /O
```

Stanford 命名实体识别不需要词性标注, 因此输出仅分为两段, 标注集也比较简单。这里标签“O”表示非专名, “LOC”表示地名。其他标签会在后面章节中再做讲解。

1.6 整合句法解析模块

句法分析是根据给定的语法体系自动推导出句子的语法结构, 分析句子所包含的语法单元和这些语法单元之间的关系, 将句子转化为一棵结构化的语法树。句法分析是所有自然语言处理的核心模块(一般拼音文字没有分词的问题)。其研究历史比较悠久, 应用也很广泛。

目前句法分析有两种不同的理论: 一种是短语结构语法; 另一种是依存语法。关于两种理论更详细的内容在后面章节会详细分析, 这里先不做讨论。句法分析的开源系统也很多, 但迄今为止, 这些解析技术都还不够理想, 仍旧很难找到高精度处理中文的句法解析系统。

其中, 比较突出的是 Ltp 3.3 中文句法分析系统, 使用依存句法理论。在 SANCL 2012 互联网数据依存句法分析评测中获得第二名; 在 CoNLL 2009 句法和语义依存分析评测中, 中文依存句法分析获得第三名。

除此之外, 最著名的句法解析器是 Stanford 句法解析器。截至 2015 年, Stanford 的句法树包含了如下三大主要解析器。

- ❑ PCFG 概率解析器。是一个高度优化的词汇化 PCFG 依存解析器。该解析器使用 A*算法, 是一个随机上下文无关文法解析器。除英语之外, 该解析器还包含一个中文版本, 使用宾州中文树库训练。解析器的输出格式包含依存关系输出和短语结构树输出。

- ❑ **Shift-Reduce 解析器。**为了提高 PCFG 概率解析器的性能, Stanford 提供了一个基于移进-归约算法 (Shift-Reduce) 的高性能解析器。其性能远高于任何 PCFG 解析器, 而且精度上比其他任何版本 (包括 RNN) 的解析器都更准确。
- ❑ **神经网络依存解析器。**神经网络依存解析器是深度学习算法在句法解析中的一个重要应用。它通过中心词和修饰词之间的依存关系来构建出句子的句法树。有关此方面的研究是目前 NLP 的研究重点。

本节延续 1.4 节的系统, 使用 Ltp 3.3 和 Stanford Parser 来进行中文的文本解析。因为 Stanford 的句法解析器比较多, 这里仅实现比较有代表性的 PCFG 解析器。

1.6.1 Ltp 3.3 句法依存树

继续使用 Ltp 3.3 中文句法解析模块。图 1.5 列出的语言包列表中, 句法解析模块的文件名为 parser.model。

为了简化, 下面使用已经分好词的单句进行演示。

代码如下。

```
# -*- coding: utf-8 -*-
import sys
import os
import nltk
from nltk.tree import Tree # 导入 nltk tree 结构。
from nltk.grammar import DependencyGrammar # 导入依存句法包
from nltk.parse import *
from pyltp import * # 导入 ltp 应用包
import re

reload(sys)
sys.setdefaultencoding('utf-8') # 设置 UTF-8 输出环境

words = "罗马尼亚 的 首都 是 布加勒斯特。".split(" ") # 例句

postagger = Postagger() # 首先对句子进行词性标注
postagger.load("X:\\ltp3.3\\pos.model")
postags = postagger.postag(words)

parser = Parser() # 将词性标注和分词结果都加入分析器中进行句法解析
parser.load("X:\\ltp3.3\\parser.model")
```



```
arcs = parser.parse(words, postags)
arclen = len(arcs)
conll = ""
for i in xrange(arclen): # 构建 Conll 标准的数据结构
    if arcs[i].head == 0:
        arcs[i].relation = "ROOT"
    conll += "\t"+words[i]+"("+postags[i]+")"+" \t"+postags[i]+" \t"+str(arcs[i].
head)+" \t"+arcs[i].relation+"\n"
print conll
conlltree = DependencyGraph(conll) # 转换为依存句法图
tree = conlltree.tree() # 构建树结构
tree.draw() # 显示输出的树
```

输出结果如下。

罗马尼亚(ns)	ns	3	ATT
的(u)	u	1	RAD
首都(n)	n	4	SBV
是(v)	v	0	ROOT
布加勒斯特(ns)	ns	4	VOB
。(wp)	wp	4	WP

输出图如图 1.16 所示。

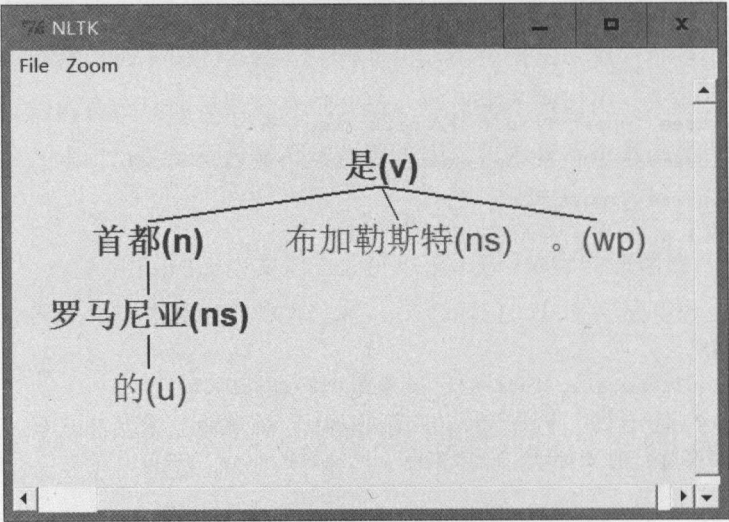


图 1.16 Ltp 3.3 句法依存解析树

如图 1.16 所示，结合两个输出结果，将单句“罗马尼亚 的 首都 是 布加勒斯特”的解析结果给出如下结论：句法树是一棵依存关系树，根节点为其谓语动词“是”，主语是“首都”，“罗马尼亚”是修饰“首都”的定语，句子的宾语是“布加勒斯特”。

1.6.2 Stanford Parser 类

如果仅使用斯坦福的中文句法解析模块（也称为 Parser），可从 <http://nlp.stanford.edu/software/lex-parser.shtml> 下载。与命名实体识别相同，Parser 模块的程序和中文模型库也分开存放，但都在一处下载。应用程序和模型都可从图 1.17 所示的截图处下载。

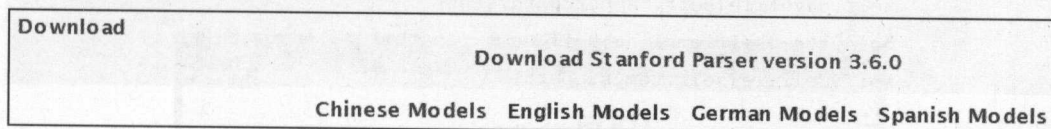


图 1.17 截图

这里仍旧使用 stanford-corenlp 中的模型和程序。在图 1.14 目录列表中对不同句法解析器有不同的模型目录，分别为：lexparser、parser 和 srparser。其中，PCFG 概率解析器模型为 lexparser；Shift-Reduce 解析器模型为 srparser；神经网络依存解析器模型为 parser。以 lexparser 为例，图 1.18 所示的 5 个模型分别用于 lexparser 解析器的调用，最常用的库是 chinesePCFG.ser.gz 文件。该文件支持高度精确的词汇解析器。

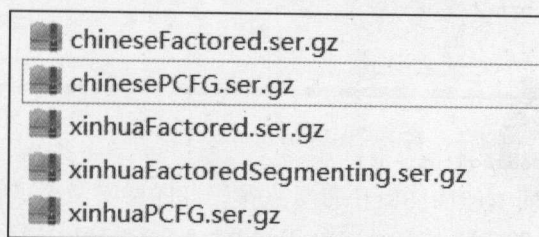


图 1.18 Stanford 句法依存模型文件

最后，仿照前两节，编写一个 Python 的 StanfordParser 接口类如下。

```
class StanfordParser(StanfordCoreNLP):
    def __init__(self, modelpath, jarpath, opttype):
        StanfordCoreNLP.__init__(self, jarpath)
        self.modelpath = modelpath # 模型文件路径
        self.classfier = "edu.stanford.nlp.parser.lexparser.LexicalizedParser"
        self.opttype = opttype
        self.__buildcmd()
    # 构建命令行
    def __buildcmd(self):
        self.cmdline = 'java -mx500m -cp "' + self.jarpath + '" ' + self.classfier + ' '
        self.cmdline += '-outputFormat "' + self.opttype + '" ' + self.modelpath + ' '
    # 解析句子
    def parse(self, sent):
```

```

        self.savefile(self.tempsrcpath,sent)
        tagtxt = os.popen(self.cmdline+self.tempsrcpath,"r").read() # 输出到变量中
        self.delfile(self.tempsrcpath)
        return tagtxt
# 输出到文件
def tagfile(self,sent,outpath):
    self.savefile(self.tempsrcpath,sent)
    os.system(self.cmdline+self.tempsrcpath+' > '+outpath )
    self.delfile(self.tempsrcpath)

```

1.6.3 Stanford 短语结构树

以短语结构的方式输出，内容如下。

```

# -*- coding: utf-8 -*-
import sys
import os
from nltk.tree import Tree # 导入nltk库
from stanford import *

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')
# 配置环境变量
os.environ['JAVA_HOME'] = 'D:\\Java7\\jdk1.8.0_65\\bin\\java.exe'
# 安装库
root = "E:/nltk_data/stanford-corenlp/"
modelpath= root+'models/lexparser/chinesePCFG.ser.gz'
opttype = 'penn' # 宾州树库格式
parser = StanfordParser(modelpath,root,opttype)
result = parser.parse("罗马尼亚 的 首都 是 布加勒斯特 。")
print result
tree = Tree.fromstring(result)
tree.draw()

```

输出结果如下。

```

(ROOT
  (IP
    (NP
      (DNP
        (NP (NR 罗马尼亚))

```



```

(DEG 的))
(NP (NN 首都)))
(VP (VC 是)
  (NP (NR 布加勒斯特)))
(PU 。)))

```

输出图 1.19。

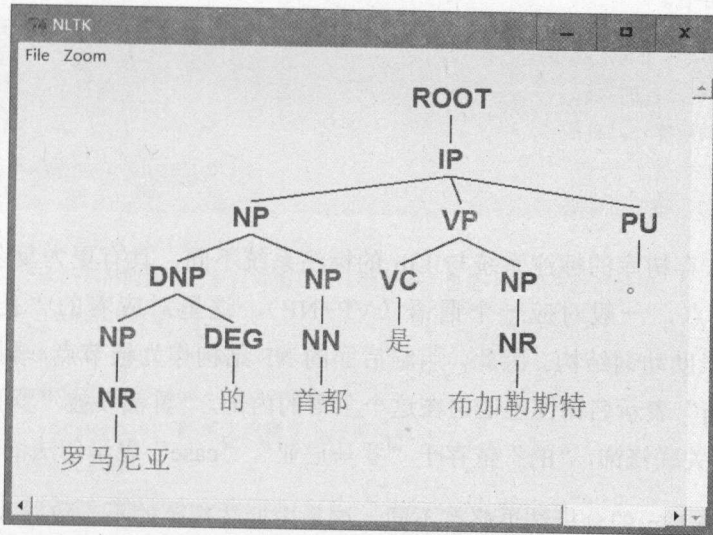


图 1.19 Stanford 句法结构树

如图 1.19 所示，短语结构树的叶子节点是每个词的词形，词形上一级节点是词性，再上一级节点是由多个词构成的短语组块，“NP”这里表示名词性短语，“DNP”是由 NP+“的”构成的短语结构，“VP”是动词短语，“IP”是简单句。

1.6.4 Stanford 依存句法树

以依存句法的方式输出，内容如下。

```

# -*- coding: utf-8 -*-
import sys
import os
from nltk.tree import Tree
from stanford import *

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

```

```

root = "E:/nltk_data/stanford-corenlp/"
modelpath= root+'models/lexparser/chinesePCFG.ser.gz'
opttype = 'typedDependencies' # "penn,typedDependencies"
parser=StanfordParser(modelpath,root,opttype)
result = parser.parse("罗马尼亚 的 首都 是 布加勒斯特 。")
print result

```

输出结果如下。

```

assmod(首都-3, 罗马尼亚-1)
case(罗马尼亚-1, 的-2)
nsubj(布加勒斯特-5, 首都-3)
cop(布加勒斯特-5, 是-4)
root(ROOT-0, 布加勒斯特-5)

```

Stanford 依存树库的标注系统与 Ltp 的标注系统不同，具有更为复杂的依存结构。“root”是根节点，一般对应一个谓语（VP+NP），这里对应着的“是”，它是一个“cop”——系表助动词结构。因此，用它后面的 NP 结构作为根节点，剩下的部分就很容易了。“nsubj”表示名词性主语，在这个主语的内部，“首都”被“罗马尼亚”修饰，“assmod”表示关联修饰，“的”依存于“罗马尼亚”，“case”表示句法依赖。

这个结果与 Ltp 的分析结果略有不同，这是由标注规范的不一致导致的。有关标注规范的更多细节，第 6 章将给出具体说明。

1.7 整合语义角色标注模块

语义角色标注（Semantic Role Labeling, SRL）来源于 20 世纪 60 年代美国语言学家菲尔墨（C.J.Fillmore）提出的格语法理论。在菲尔墨看来，格关系是句子深层结构中的名词和谓语动词之间的一种固定不变的语义结构关系（谓词—论元关系），而这种关系和具体语言中的表层结构上的语法结构没有一一对应关系。换句话说，有同样谓词—论元结构支配的，但语法结构不同的句子，其语义应该是相同的。

该理论是在句子语义理解上的一个重要突破。基于此理论，语义角色标注就发展起来了，并成为句子语义分析的一种重要方式。它采用“谓词—论元角色”的结构形式，标注句法成分相对于给定谓语动词的语义角色，每个语义角色被赋予一定的语义。

美国宾州大学已经开发出一个具有实用价值的表示语义命题库，称为 PropBank。在本书的后续章节将会对该主题库做一个全面和深入的分析。

语义角色标注系统已经处于 NLP 系统的末端,其精度和效率都受到前面几个模块的影响,所以,当前系统的精度都不高,在中文领域还没有投入商业应用的成功案例,本节介绍的是 Ltp 3.3 中文语义角色标注系统。

```
# -*- coding: utf-8 -*-
import sys
import os
from pyltp import *

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

MODELDIR = "E:/nltk_data/ltp3.3/"
sentence = "欧洲东部的罗马尼亚,首都是布加勒斯特,也是一座世界性的城市。"
segmentor = Segmentor()
segmentor.load(os.path.join(MODELDIR, "cws.model"))
words = segmentor.segment(sentence)
wordlist = list(words) # 从生成器变为列表元素
postagger = Postagger()
postagger.load(os.path.join(MODELDIR, "pos.model"))
postags = postagger.postag(words)

parser = Parser()
parser.load(os.path.join(MODELDIR, "parser.model"))
arcs = parser.parse(words, postags)

recognizer = NamedEntityRecognizer()
recognizer.load(os.path.join(MODELDIR, "ner.model"))
netags = recognizer.recognize(words, postags)
# 语义角色标注
labeller = SementicRoleLabeller()
labeller.load(os.path.join(MODELDIR, "srl/"))
roles = labeller.label(words, postags, netags, arcs)
# 输出标注结果
for role in roles:
    print 'rel:',wordlist[role.index] # 谓词
    for arg in role.arguments:
        if arg.range.start!=arg.range.end:
            print arg.name,' '.join(wordlist[arg.range.start:arg.range.end])
        else:
            print arg.name,wordlist[arg.range.start]
```


输出结果如下。

```
rel: 是
A0 欧洲 东部 的 罗马尼亚
A0 首都
A1 布加勒斯特
rel: 是
ADV 也
A1 一座 世界性 的
[INFO] 2016-01-09 02:40:34 build-config: add 61 constrains.
[INFO] 2016-01-09 02:40:34 report: number of labels 13
```

这里“rel”标签表示的是谓词，“A0”指动作的施事，“A1”指动作的受事。关于其他标签，在后面会专门讲解。

1.8 结语

作为全书的开篇，本章的内容可能有些繁杂。为此，下面总结一下本章的主要内容。首先，本章回顾了自然语言处理的简要历史。自然语言处理及与之相伴的智能计算，从诞生到现在，大约经历了4个阶段，分别是诞生前的科幻主义时代、基于图灵机的早期探索、中期规则派和统计派的划分，以及现在的认知计算时代。

在1.2节给出了现代自然语言处理系统的基本架构，即从中文分词、词性标注到句法解析的语法分析流程，以及与之并行的命名实体识别、语义组块、语义角色标注的语义分析模块。整个框架从中文分词开始直到语义角色标注为止，共历经4~5个串联的环节。后面各节就是对上述环节的介绍和基本应用。

注意，这是当代NLP流行框架，并非本书提出的NLP整体架构。本书的NLP整体架构将在后面章节中给出。

基本上，现代的自然语言处理任务和应用系统都依据该框架发展开来。从1.3节开始，结合StanfordNLP和Ltp 3.3两大开源系统，以及相关的一些小型NLP系统，逐一分析总体架构中各个模块的功能

在中文分词中，介绍了Ltp 3.3和结巴分词的中文分词模块的应用；在词性标注一节中，给出了Ltp 3.3和StanfordNLP中的词性标注模块的应用；在命名实体识别一节中，给出了Ltp 3.3和StanfordNLP中的命名实体识别模块的应用；在句法解析一节中，给出

了 Ltp 3.3 的依存解析树, 以及 StanfordNLP 的短语结构解析树和依存句法解析树的应用。最后, 在语义角色标注系统中, 给出了 Ltp 3.3 的语义角色标注系统的应用。

各 NLP 模块均包含如下结构。

- ❑ 模块功能概述。
- ❑ 资源下载地址。
- ❑ 应用执行代码。
- ❑ 执行结果及其说明。

以这些著名的开源系统作为本书的开端, 其目的是为读者提供一个开放的学习平台。本书仅仅起到抛砖引玉的作用, 读者在阅读本书后, 通过学习这些著名的 NLP 系统的更多规范、算法, 可以掌握语言的基本理论、NLP 的设计思想, 以及开发 NLP 的各种算法, 最终有能力构建出自己的 NLP 系统。

第 2 章

汉语语言学研究回顾

严格意义上讲，本书是计算机行业、人工智能领域的技术类书籍。其主要内容也以程序和算法设计为中心。由于研究领域的特殊性，我们需要对研究对象——汉语语言有一个较为深入的认识，才能更好地理解后面各类算法的意义。本书简要介绍了汉语的发展历程，以及各个阶段的主要事件及变革。大部分内容都与自然语言处理有直接或间接的关系。更具体地说，与汉语的各个部分，如字、词、句子的内部结构和模式相关，而语言学方面的其他知识，本书尽量不涉及或少涉及。

汉语属于汉藏语系，与世界各国广泛使用的拼音文字相比，它更像一种古老的孤立语。无论从字符的结构和形式上都显得特立独行。这是中华民族独特的地理位置和长期统一的发展历程所决定的。虽然，汉语在历史上先后吸收和同化了匈奴、鲜卑、突厥、契丹、满、蒙古、梵语等语言中的许多成分，但是两千多年来，汉语特有的符号化表现形式却一直没有改变过。

2.1 文字符号的起源






简要地回顾文字起源，将有助于读者对研究对象形成完整的认识。只有了解历史，才能正确地理解现在、准确地预见未来。

有关文字符号的起源，虽然时隔久远，现代各国的语言学家和考古学家都研究得已



经比较充分了,并形成了一致的看法。在人类早期的社会生活中,由于对信息交流、情感交流和思维交流的需要,逐渐形成了语言。最初的文字符号也称为记事符号,几乎与语言形成于同一时期,但完整的文字系统应形成于语言之后。因为文字突破了语言的时空范围,是凝固的语言,所以文字并非从一开始就记录人们语言中的所有内容,而是有选择地记录一些对人们生产、生活比较重要的部分。千百年来,随着人们生产、生活的不断发展,文字也在不断发展,并且逐渐促进了语言进一步的统一和规范。这是人类语言发展的共同规律。

2.1.1 从记事谈起

大量的考古事实证明,文字的产生从氏族公社的记事开始。最初的文字符号记录了氏族群体的劳动和分配、祭祀和占卜等活动。在系统的文字符号形成之前,这些氏族成员采取以结绳记事为主的各种手段,最常用的主要有三种:结绳记事、岩画记事和刻契记事。这三类记事方式形成了后来文字符号的主要来源。

结绳记事起源于母系氏族社会——旧石器时代后期。当时绳索是原始人生活中的重要发明和重要用品,不但用来捆束东西、捆扎武器、捆绑野兽、遮盖妆饰身体,而且用来记录生活中的大事,特别是与数字相关的大事,如劳动所得的分配计数等。“记”字在古代应写为“纪”。甲骨文中“己”是“纪”的本字。己,甲骨文、金文像丝绳缠绕绑扎的样子。当“己”的“结绳记事”本义消失后,篆文在的基础上再加“丝”另造“纪”代替,强调用丝绳打结作记号(象形词典:<http://www.vividict.com/WordInfo.aspx?id=1723>)。

牟作武在《中国古文字的起源》一书中写道:

在古代的甲骨文字中的数字还保留着结绳记数的形象(见图2.1),如横列的结绳表示一条为“一”,二条为“二”,三条为“三”,四条为“四”,交叉结绳为五;竖列的结绳,一条为“一十”,二条为“二十”,多位数的排列跟后来的苏州码相近(见图2.2)。以上所说仅是今天能以考断的极少数的痕迹,就足以说明结绳记事的情况。但应指出,人类的进化,各人种、各民族、各个部落,在一个共同的规律制约之下,都有自己独特的内容和形式。结绳记事也是这样,各个部落氏族都有自己的结绳记事。在甲骨文中,和也是结绳的写照。

如图2.1所示,结绳的出现说明了一个重要的事实,在形成文字符号系统之前,原始人类已经形成相对完备的数字符号的体系,但受限于结绳工具的简陋,结绳只是原始

的记事法，常常无法表达事件的复杂性及丰富的形象，不可能独立完整地记录事件，也不能表示语言中的读音，因此不可能直接发展为复杂的汉字符号。考古发现虽然某些汉字的形体与结绳有零星的关系，但结绳记事不是汉字的主要来源。

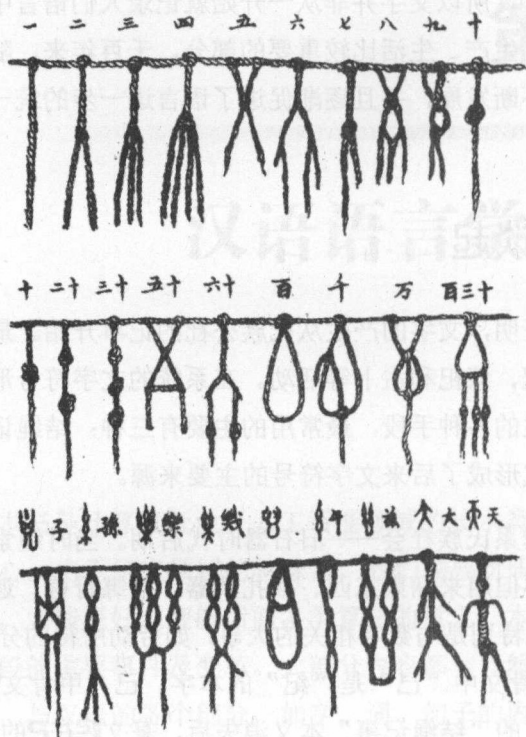


图 2.1 结绳记数（选自《中国古文字的起源》）

这样又过去了千百年，原始人又发展出了岩画记事和刻契记事。岩画可能起源很早，不能以现存的岩画为据说明它的起源。中国自古就有“书画同源”一说，这是因为最早的中文文字推测主要来源于岩画，汉字符号与岩画如同兄弟，同根而生，有很多内在的联系。

象形字来源于绘画的重要证据是东巴文。东巴文是居于西藏东部及云南省北部的少数民族纳西族所使用的文字。东巴文源于纳西族的宗教典籍兼百科全书的《东巴经》。由于这种字由东巴（智者）所掌握，故称为东巴文。

东巴文是一种原始的图画象形文字，现在仍为东巴教徒传授使用，书写东巴经文。纳西话叫“司究鲁究”，意为“木迹石迹”，见木画木，见石画石。东巴文是一种兼备表意和表音成分的图画象形文字。其文字形态十分原始，甚至比甲骨文的形态还要原始，属于文字起源的早期形态，但亦能完整记录典藏。

如图 2.2 所示,这种最古老的文字,图绘性很强,表明它们都与岩画之间有一种渊源关系,应该是由岩画发展而来的。它是世界上最古老的文字的共同特征,也是岩画记事作为文字主要来源的有力证据。汉字体系的形成,也应该经历过由图画到文字的阶段。汉字的起源就是原始的绘画。原始人在生活中用来表达生活的“图画”形式,慢慢地从原始图画变成了一种“表意符号”。



图 2.2 东巴文译注(选自《百度百科—东巴文》)

与此同时,另一种记事符号由于其使用方便,也逐渐发展起来。《中国古文字的起源》一书中写道:

原始人的刻划(刻契)记事有三种情况,一是他们把特定的记号和数标刻在兽骨、器皿上,旧石器时代我们的祖先就具有坚锐的雕刻工具。峙峪遗址中发现旧石器时代有刻划符号的马骨,贵州兴义县发现有刻记的象牙。

.....

三是氏族和部落出现了专门记事用的刻划木桩(见图 2.3)。刻划记事的木桩发展到后来便成了复杂的综合记事的“记事桩”。记事桩将刻划记事、系物记事、结绳记事集于一体,并出现了分类记事的手段,如狩猎、战争和部落氏族内的物质分配、祭祀、占 A 分类分组记录。不过记事和记数采取不同的手段。例如,记事利用系物的办法,记数采取结绳办法,记时则用刻划(刻契)的办法。这些办法综合在一起,制作一个庞大的记事桩。

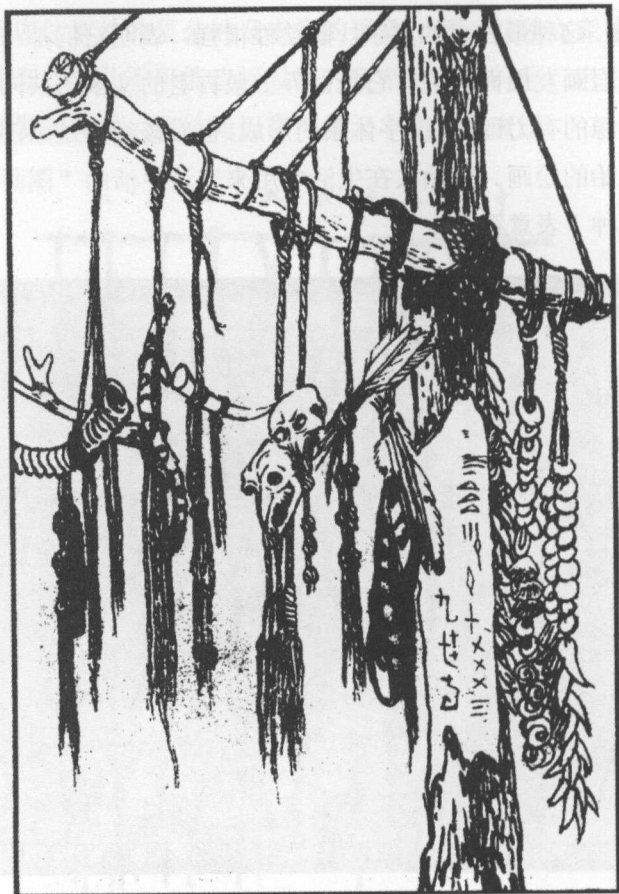


图 2.3 记事木桩（选自《中国古文字的起源》）

如图 2.3 所示，三种记事方式可能在远古时期相继出现并同时存在；同是记事功能，但使用的分域不同，结绳记事多用于物质分配，记录战事，渔猎的次数等，偏于量的记载。而岩画则偏重于那些值得永久性纪念的渔猎奇迹、大型庆典、重大事件的场景和过程。刻契记事最初用来计时，广义上来讲，它也是一种记数的手段，后来逐渐发展成记录各种记事符号的工具。几种刻契记事记数的实例如下。

青海省乐都县柳湾原始社会末期墓地出土的 40 件带刻口的骨片（见图 2.4）。

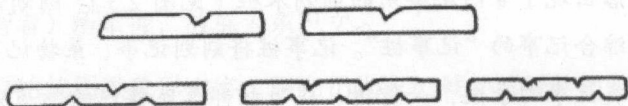


图 2.4 柳湾原始社会末期墓地出土的带刻口的骨片

云南福贡地区的傈僳族、云南澜沧拉祜族等少数民族地区，直到 20 世纪 50 年代，仍有记数习俗惯制在民间流行（见图 2.5）。

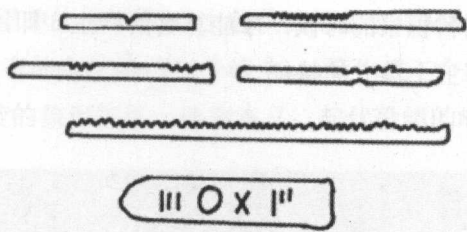


图 2.5 几个少数民族刻契记数形制

西安半坡文化遗址出土的彩陶上也有记数的刻契符号（见图 2.6）。

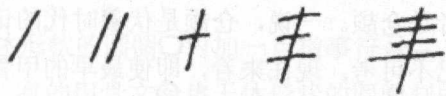


图 2.6 西安半坡文化遗址出土的彩陶上的刻契符号

如图 2.4~图 2.6 所示，新石器时代的陶器上刻有大量类似文字的符号，表明人类记事方式的进步。我们现在所能见到的与文字起源有关的实物资料是陶片上的刻画符号，称为“陶文”。图 2.6 为半坡、马家窑、姜寨等仰韶文化遗址中发现的陶文。于省吾先生考证：“五作×，七作+，十作|，二十作||，示作T，矛作↑等……这就是文字起源阶段所产生的一些简单文字。”

刻契记事替代结绳记事是一种书写载体的革新，它使记事载体从一维发展到了二维。刻契记事革新了记事符号的载体，这就使记事符号本身得到了进一步发展。刻契还从岩画中吸取和抽象了大量的图形符号，抽象出事物的形象，以此丰富了记事内容。因此，刻契记事为各类记事符号转变为抽象的文字符号提供了介质。

有证据表明，刻契记事后来直接发展出了两河流域（底格里斯河和幼发拉底河流域）苏美尔人使用的楔形文字。

2.1.2 古文字的形成

在漫长而缓慢发展的原始社会后期，记事符号逐渐演变为了文字符号。当社会进入到奴隶制时代，氏族公社的组织形式逐渐正式化，最终演变成国家，文字就逐渐产生了。由于历史久远，迄今所能见到的、真正形成体系的最早汉字是商代的甲骨文，但据实际情况来看，汉字体系的形成，可能在更早的年代——夏王朝时期。

这里解释一下何为形成体系的概念。文字是一种记录语言的符号，原始的记事符号必须脱离了任意绘形、任意理解的阶段，产生一批具有约定的意义，具有固定读音的单

字，并且可以开始进行语料积累的时候，才能算真的产生。即，文字区别于图绘或其他
的记事符号必须具有如下三个独立的特征。

- ☐ 约定的意义。
- ☐ 基本固定的读音。
- ☐ 基本一致的形式。

简单地讲，文字符号必须具备形、音、义的三者一致性。这就必须对当时的记事符号有一个再创造的过程，历史上称这个过程为造字。

传说中，汉字的创造者是仓颉。一说，仓颉是伏羲时代的记事官；一说，仓颉是皇帝时代的史官。历史久远已不可考。现在来看，即使最早的甲骨文汉字，也绝不是一个
人能够完成的，必然经历了漫长的积累和演变的过程。应该讲，原始造字时期的文字创造，
是氏族中的记事人员（巫蛊）在氏族酋长的认同下，与氏族成员共同议定的结果，
可以说是长时间集体积累和创作的产物，并在长时间的氏族扩张和衰亡的征战中不断传
承与湮灭。

但是，文字自产生开始便有两种矛盾始终与之伴随：一种矛盾是文字与语言的矛盾，
即文字的数量多寡与能否表达与之对应的语言之间的矛盾，这个矛盾是第一位的，下面
要讲的“六书”造字法就是为了解决这个矛盾，而创造出来的汉字造字法；另一种矛盾
是文字与书写者的矛盾，是第二位的，这一矛盾的解决在 2.3 节中具体介绍。

2.2 六书及其他

后人在总结前人的造字方法时，最重要的成果就是六书。它是由汉代学者根据汉字的
构成和使用方式归纳成的六种模式，总称为六书。六书是指：“象形”、“指事”、“会意”、
“转注”、“假借”、“形声”。下面以许慎的《说文解字》作为根据，结合象形词典
(<http://www.vividict.com/>) 给出六种模式在构成、认知方面的解读。

2.2.1 象形

“象形者，画成其物，随体诘屈，日月是也。”这句话说的是，所谓的象形字是把具
体的物体以绘画的形式表现出来，形成文字，根据物体的不同绘画形式也不同，如图
2.7 和图 2.8 所示为象形字。






















山，甲骨文像遥望中的地平线上起伏连绵的群峰的线描，有三（众多）座峰头。金文写成剪影。有的金文将三个峰头简化成三个短竖，淡化峰尖形象。篆文保留中间一座峰岭的象形特征。造字本义：起伏叠嶂的峰岭。



图 2.7 “山”的字形演变

日，甲骨文在天体形状的圆圈内加一点指事符号，表示发光特性的天体。由于甲骨文刻画得不流畅，有的甲骨文将天体形状的圆圈刻成五边形；有的甲骨文将圆圈刻成棱形。造字本义：在太空运行、发光的天体，太阳。金文承续甲骨文字形。篆文将金文字形中的圆点指事符号写成短横，并与方框连接，使字形抽象化。

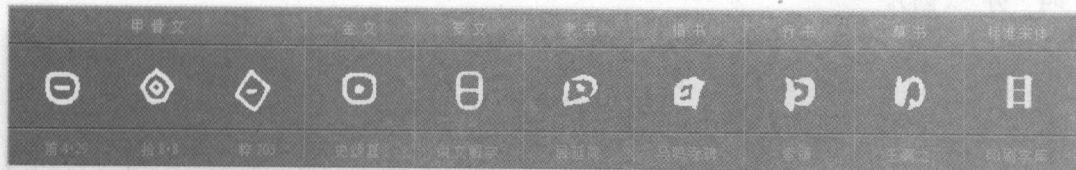


图 2.8 “日”的字形演变

世界各类古文明的文字大多起源于象形字。受限于当时使用的书写工具，用文字的线条或笔画把要表达物体的外形特征简单地勾画出来。象形文字抽象掉了自然界事物的具体图形（颜色、明暗等特征），仅保留了形状上的相似。象形字的形体直观反映了自然界的客观事物本身。

虽然象形字的原理简单，但是意义却很深刻。一方面，它是早期人类对客观世界的一种最基本的编码方式，然而这一方式却包含两种重要的机制：模仿和抽象。模仿和抽象都是重要的认知现象，而抽象在认知层面要高于模仿。简单而言，模仿可以来源于经年累月的观察和学习，只要重复的次数足够多，人人都可以掌握。模仿机制直接形成了早期的岩画，但抽象则不同，抽象具有高级思维的某些特征。例如，鱼可以分为鲤鱼或鲫鱼，还可以分为大鱼和小鱼，它们的外观或尺寸上有明显的差别，但从一个抽象的高度的来看，它们都在水中生活、身体都呈纺锤形、有鱼鳍、鱼鳞、用鳃呼吸，都属于同类，即同属于鱼的属类（范畴），于是就用“鱼”这个象形字来统称它们。早期的人类通过构造象形字建立了属类的思维。文字的产生，首先是象形字的出现，使原始人类发展出了

范畴的概念——一种认知世界的基本能力。

另一方面，它构成了后续几种造字方法的字根，也称为本字。也就是说，其他造字方法所造出的字，都以象形字的字形为基础，在此结构上做出某种变形。有的添加笔画，反映了局部与总体的关系——指事字；有的增加偏旁部首，反映了含义与读音的关系——形声字；有的干脆就用本字表达其他的含义——假借；还有的将多个独体字组合到一起，引申出其他的含义——会意，等等。

2.2.2 指事

“指事者，视而可识，察而见意，上下是也。”这句话说的是，一眼看上去就可以识别出整体（本字），仔细观察就能发现意义所在。图 2.9 和图 2.10 所示为指事字。










刃，甲骨文是指事字，在“刀”的锋面上方加一点指事符号，表示刀口。造字本义：刀的锋利部位。篆文将甲骨文的写成。刀的锐利部位叫“刃”，用刀刃砍斫叫“刃”（创）。



图 2.9 “刃”的字形演变

“上”是特殊指事字，由两横构成，底端一横较长，顶端的一横较短。古人用代表混沌太初状态；用（二，由两个“一”组成，两横一样长）代表从混沌太初中分化出来的、相并列的天与地。古人调整表示天与地、等长的两横，以短横方向表示朝天、或朝地的方向。造字本义：与地相对的天。现代的意义表示在当前位置的上面。

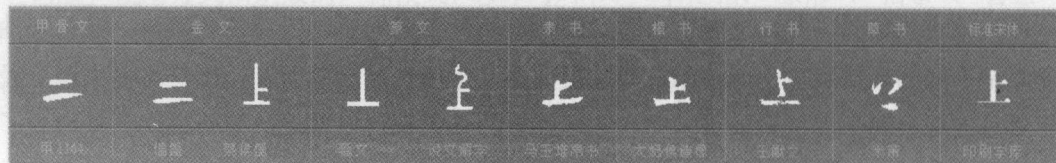








图 2.10 “上”的字形演变

在人们的生产生活中，有时候需要强调地不是整体，而是局部，或者需要参照已知位置指示出其他的位置，那么怎么办呢？很简单，就是在象形本字的相应部位加上一个标识符号，以指示所表示的局部范围或相对位置。

如上例,“刀”锋利的部分称为“刃”;“天”相对于“地”为“上”。这就是该造字法的基本含义。这里“刀”和“一”是本字,“丶”和“丨”为指事的符号。指事字的含义很清楚,它通常表示某种整体与局部的关系或者相对位置的概念。指事造字法说明原始人在使用文字过程中,逐渐形成了事物的局部和总体认识,以及位置上相对的认识。基本上,最初的指事字,其本字都是象形字。














2.2.3 会意

会意:会意者,比类合谊,以见指撝,武、信是也。这个造字法,是将两个或两个以上的字根组合起来,使之形成一个新字,并衍生出新的含意。图 2.11 和图 2.12 所示为会意字。

木,既是声旁也是形旁,表示树。休,甲骨文 (人) +  (木),像一个人 靠在大树 的枝叶之下,表示古人在野外劳作时,选择能遮阳蔽雨的树下歇息。造字本义:在树荫下乘凉歇息。金文、篆文 承续甲骨文字形。“休”指停止肢体劳顿。

甲骨文	金文	篆文	隶书	楷书	行书	草书	标准宋体
 							
前3·26 康上12·7	杨鼎	说文解字	范石碑	颜真卿	陈师楷	徐伯清	印刷字库

图 2.11 “休”的字形演变

盗,甲骨文 (欠,表示不足,引申为贪欲) +  (水,一般是河流,引申为边界) +  (舟,表示乘船越界)。造字本义:出于贪欲,过河越界,劫物掠货。篆文 将金文的“舟” 写成“皿”。隶书 将篆文的“水” 简写成“两点水”;将篆文的 写成;将篆文的 写成。








甲骨文	金文	篆文	隶书	楷书	行书	草书	标准宋体
							
前六·三二·五	智缺	说文解字	何籀	智缺	董道全	蒋谷进	印刷字库

图 2.12 “盗”的字形演变

会意造字常用两个及两个以上的独汉字,根据各自的含义通过左右拼接、上下拼接等方式表示更为复杂的含义,从而构造出新字。其表示的含义也逐渐脱离了直观的自然界事物,而引申为表达人们生产生活中的某种关系或活动,常用来表示某种行为或状态。

例如，“休”字表示倚靠着“树”的“人”，说明此人正处于休息状态；“盗”字表示“因为收入不足，产生贪欲，越过边界，抢夺他人的财务”，这种行为称为“盗”。这些字反映了原始人类生产、生活活动的方方面面。

就本文研究的范畴而言，我们更注重字的含义与结构的关系。一般会意字的字形由多个独体字并列构成——这里的并列是指意义上的并列，其中每个独体字都有独立的含义，会意字的整体含义由各个独体字共同构成，缺少了哪一部分，都无法正确理解该字的意义。另一方面，会意字中的独体字之间有位置、大小的差异，不同的位置甚至形式，都会对会意字的含义产生影响，例如“武”字，从戈从止。止是趾本字，戈下有脚，表示人拿着武器走，有征伐或显示武力的意思。再如“从”字是一个人跟着另一个人走，表示跟随。而“比”，表示两人接近并立。

会意字的产生反映了人类一种更高级的认知形式，它通过两个或多个事物的组合关系，派生出了新的意义。我们需要研究和理解这个派生的过程。以上文中的“盗”为例，如图 2.13 所示，该字最初描述了一个过程，“常因为不足，贪欲，过河越界、劫物掠货”。之后该含义被泛化为“一切用不正当的手段谋取他人财物的行为”。第一次引申（扩大引申）之后，“过不过河，是不是因为收入不足”这些具体的行为原因就显得不重要了，而保留并强调了“用不正当手段谋取他人财物”的行为——表示动作。再向后引申（词性引申），还可以指实施此类行为的“人”——强盗、盗贼。

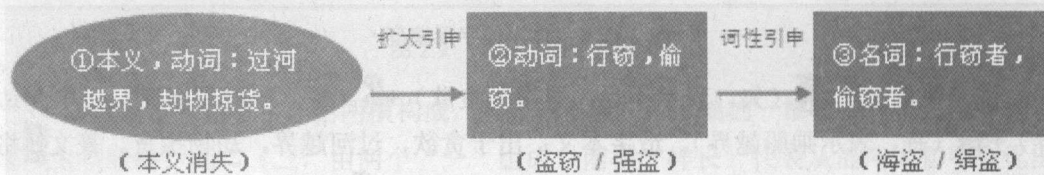


图 2.13 “盗”语义引申

这里所说的引申大体上可以分成隐喻和换喻两种方式。有关隐喻或换喻的具体分析，我们放到后面的章节中再做解释，本节以隐喻为例来解释引申的来源和基本含义。

文学上，隐喻被看作一种修辞方式，用一种事物来比喻另一种事物，目的是使行文更加生动、更有感染力。例如，我爱北京——祖国的心脏。此句中北京被拟人化为祖国的心脏。但这里，我们所说的隐喻和换喻不局限于语言文学中的修辞方法，而把它看作一种认知现象。那么，引申就是一种人类重要的认知能力。

为什么常把一种事物与另一种事物进行类比呢？因为，在生产、生活中，人们偶然遇到一种新生事物（或抽象事物）时，常常需要把所见、所闻、所想传达给别人，大家

形成共同的认知,以便做出决策。语言作为一种沟通交流的重要手段,必须要完成这个过程。那么,我们如何描述和说明这个新事物呢?因为其他人并没有亲眼见过此类事物,描述该事物的形体和外貌可能并不困难,但是想要进一步说明它的内在属性、行为、结构或机制就不那么简单了。

此时,就需要一种新的描述方式,将新事物的属性、行为、结构或机制提炼出来,与某种或多种特征相似的、我们熟悉的事物进行类比。相互类比的两者中,其中一种事物是有待认识的较为陌生的事物,而另一种事物则是我们较为熟悉的,很容易理解其相关特征和运作方式的事物。这就使我们把对熟悉事物的认知转移到了新事物上(或抽象事物上),完成了对新事物的认知过程。

隐喻(或说引申)是一种通过文字(或语言)来完成的高级认知模式。它简化了大量描述新事物各种特征的细节,使认知过程变得更加简单,从而加快了我们认知事物的速度。会意字从构字法上体现了这种认知模式的基础形态。

2.2.4 形声

形声:形声者,以事为名,取譬相成,江河是也。所谓以事为名,即依事类而定其名字。是说在经某个事物定名而造字时,先确定它在客观事物中的属类(范畴),属类确定后就用表示此属类的字来做新造字的语义部分;所谓取譬相成,就是根据语音取一个读音相同或相近的字来做新造字的标声部分。使用语义和标声的两个部分共同构成所造的新字。







形声法是在象形字、指事字、会意字的基础上形成的一种新的造字法,它仅由两个独体字——表示意义范畴的意符(形旁)和表示声音类别的声符(声旁)复合而成。意符一般由象形字或指事字充当,后来发展为偏旁部首的形旁;声符可以由象形字、指事字、会意字来充当,后来发展为偏旁部首的声旁。

形声法巧妙地把读音与语义结合起来,简化了造字的结构,又清晰地表达了事物的范畴和读音,不仅便于构造新字,也便于记忆。现在普遍认为,早期人类的语言形成于文字之前。一个事物往往先有读音,如果需要记录,再行造字。只要某个事物或概念有其属类(范畴),并有约定俗成的读音,就能自然地通过形声法构造出新字。因此,形声法一经出现,就成为最能产的造字形式。仅以甲骨文为例,形声字约占27%。现代汉语中的形声字已达90%以上,成为最主要的汉语造字方法。

在形声字之前，象形、指示和会意所造的字，其字形与字义是统一的，字形能够完整地表达语义。但从形声字开始，这种情况发生了变化。一部分表义的功能让位于表音的声旁，更便于将语言中表义的音节迅速构成文字。这使文字在数量上发生了质的飞跃，同时，形声造字法使字形与字义逐渐分离开来。这是汉字走向符号化的第一步。

2.2.5 转注

“转注：转注者，建类一首，同意相受，考老是也。”这句话说的是，用一个部首来表征部内的字，意义相同的字之间可以相互解释。如“考”部的考字和老字就是这样（在甲骨文中“考”和“老”字系出同源）。与此相类似的还有“母”和“女”，“帚”和“妇”，等等。以“母”、“女”为例简要说明，如图 2.14 和图 2.15 所示。

母，甲骨文在“女”的胸部位置加两点指事符号，表示妇女因生育而发达的两乳。造字本义：婴儿的生育、哺乳者。金文、篆文承续甲骨文字形。隶书有所变形。

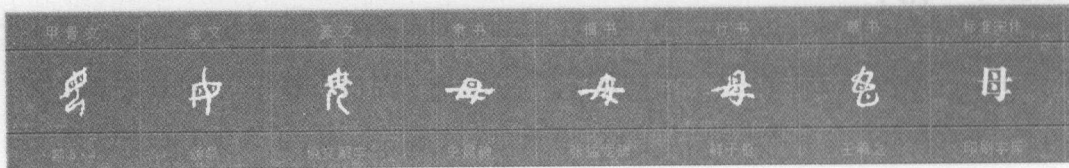












图 2.14 “母”的字形演变

女，甲骨文像一个屈膝跪坐的人娴静地交叠着双手。有的甲骨文头部位置加一横指事符号，表示发簪。造字本义：两胸饱满的妇人，能生育、哺乳幼儿的雌性。金文、承续甲骨文字形。篆文基本承续金文字形。隶书严重变形，以致“人”形消失，“手”形消失。

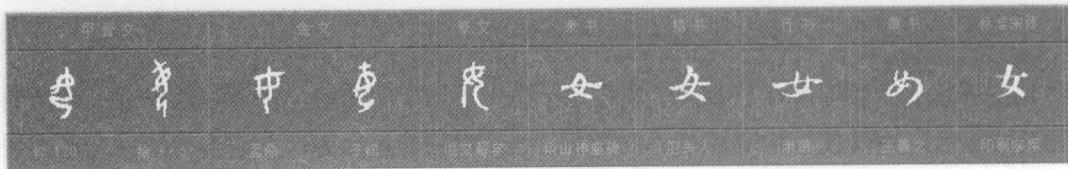


图 2.15 “女”的字形演变

转注是原始文字规范化的开始。转注暗示了这样一种规则——语义上近似的两个字，其字形也应该尽量相似，其不同之处可以通过其他造字模式来弥补。例如，“女”通过加入表示“因生育而发达的两乳”的两点，即指事造字法，构造出了新字“母”。“帚”在

甲骨文中是指由一簇干芦花用绳捆扎成的扫地工具。旁边增加一个“女”，就变成了女子在家做扫地等家务，即在家扫地做家务的女主人。很显然，新字“妇”是通过“帚”+“女”合并得到的，这是会意法。

转注法的提出说明上文所述的4种造字法并不完整，表示同一语义的字可能有很多种。例如，“妇”的语义是女主人，即使在奴隶社会，女主人也不一定在家要扫地，也可能还做饭、带孩子、洗衣服，甚至可能什么都不做，就是待着，天天和男主人吵架玩，等等。仅通过诸如象形、指事、会意这三种方法造出的女主人，可能是千差万别的，而且每个字仅仅代表了女主人这个语义的某一方面，而不能反映整体。

转注造字法的提出解决了这个问题。该方法强制将语义相近的字（词）都归为一类，使用同一或相近的字形（偏旁部首）来构造。这么做的好处是显而易见的，因为氏族公社或奴隶制国家的文字权通常都掌握在记事官（常为巫蛊）的手中，氏族公社成员或国民在部落或国家内都服从统一文字的规范。这使此种造字法的实施变得非常容易。

以殷商时期的甲骨文为例，从1899年的首次发现，共计出土甲骨154 600多片（它们分布在世界各地，其中内地收藏97 600多片，台湾地区收藏30 200多片，香港特别行政区收藏89片，总计中国共收藏127 900多片。此外，日本、加拿大、英国、美国等国家共收藏了26 700多片）。到目前为止，这些甲骨上刻有的文字符号仅有4 500多个。注意，这4 500个文字符号既是字也是词。在当时的自然条件下，仅用4500个词就可以表达丰富的社会生活，不得不说不说专注法起到了重要的作用。

2.2.6 假借

假借：假借者，本无其字，依声托事，令长是也。意思是说，假借法是文字中为表达某一新事物，本来没有表示它的字，就依据读音去找一个音同或音近的现成字来赋予其新的词义，用以表达该种事物。“令”字、“长”字就属于这类构造法。简言之就是借用已有的字，表达某一新（未命名）事物的名称。

这种造词法现在还常用。例如，外来词的中文译名都是根据外来词的发音，再找到对应的汉字，一个音节、一个音节地拼出来的。至于甲骨文的读音，应属于上古读音的范畴，与现代相去甚远，和中古也有很大区别，它们是语言学家和历史学家研究的问题，已经超出了本书的范畴。

但需要说明的是，假借法用已有的汉字去记录新词，其进步的意义在于，进一步减少了需要记忆的字符数量，这是假借的积极作用；但是，用了假借法之后，一字兼

表数意，客观上造成了一些同音同形而异义的词，使人不易掌握，这也是假借的消极作用。

在汉语自然语言处理时，一词多义的现象经常出现，对于一词多义问题的处理，也是自然语言处理的一个重要的范畴。

本书并非汉语考古类的书籍，也不是语言研究类文献。这里花了很多笔墨来介绍汉字的来源和六书造字法的目的在于，从一个侧面使读者了解古人通过构造文字如何形成认知思维，以及汉语认知的特殊模式。这对于今后研究和解析词汇及文本都具有一定的意义。

2.3 字形的流变

文字一经形成就逐渐发展开来。殷商时期，文字的持有者主要是奴隶主贵族，以巫蛊为主，文字逐渐在奴隶制国家内部的统治阶层之间学习和传播，用于祭祀占卜、谋议国事、训导言词、施政文告、发号施令、鼓励士气等，成为记述帝王和贵族言行的重要记事工具。

随着应用范围的逐渐扩大，文字的用途也得到了发展，如果说《春秋》、《论语》继承了上古文字的职能，而《诗经》则完全是一部崭新的文学艺术类作品，无论从体裁到内容都是一种创新。到了春秋战国时期，文字首先在史学学者和文化领域得到广泛传播，文学体裁也不断发展，形成了著名的诸子百家、百家争鸣的时代。

但是，到了战国末期，由于连年的战争，把战时信息传播的及时性、数量和准确程度都提高到了前所未有的高度，文字的使用者进一步扩展到了一般军事统帅和普通官吏。一方面使用文字的人员的数量空前增加了，更重要的是，文字需要表达的内容已经渗透到政治生活和军事行动中，文字本身的数量也空前地扩展了。这一切都导致篆书这种绘图式（易产生歧义）的、书写缓慢、字形难以统一的象形体越来越不能满足战争和统治的需要。一场文字改革迫在眉睫。

历史上称这次重要的文字变革为“隶变”。

2.3.1 笔与墨的形成与变革

在谈隶变之前，先插入一个小插曲，也就是书写工具的变革与成熟。

现代考古学发现,笔和墨都发端于新时期时代晚期,成形于商周,发展于秦汉。后代不同时期虽有不同程度的变革,但程度远不如秦汉。在仰韶文化的遗址中,发现了许多彩绘陶器,上面所绘的图案,清晰流畅,粗细得宜,色彩和水分饱和。这不是用一般竹木削成的笔所能表现出来的,而必须用蓄水多、柔软而有弹性的裹束起来的毛才能做到。毛笔的雏形就在这个时候出现了。其实也很简单,毛笔在作为文字书写工具之前,一定用于绘画。不仅在中国,即使在世界各国的水彩绘画和油画中仍旧使用类似毛笔结构的画笔。

毛笔作为书写工具从绘画中分离出来,则始于秦代。最早的书写作毛笔,大约可追溯到2000多年前。相传有“蒙恬始作秦笔”之说。当时,秦国大将蒙恬带兵在外作战,都要定期写战报呈送秦王。为了书写方便,当时都用竹签把战报写在绫帛上。竹签不吸水,没写几笔就要蘸墨,又极易弄脏绫帛,很不方便。情急之下,蒙恬换成麻做的枪缨捆在竹签上,蘸墨在绫帛上写。这样吸墨的问题就得到解决了,但是麻的纤维比较粗,写不了小字,也勾勒不出线条。后又换成野兔尾部的峰毛。动物皮毛纤维较细,既可以用来写较大的笔画,也可以用来勾勒出较小的线条,完全符合撰写的要求。可是兽毛富含油脂,吸墨不畅。在一次偶然的机会中,蒙恬发现石灰可以去掉动物油脂,然后将自制的动物毛笔放到石灰碱性水中浸泡,兔毛的油脂去掉了,变得柔顺起来。这样,书写用笔就产生了。

毛笔来历的传说,其真实性已不可考,但从故事中我们能够体会到毛笔的形成并非一帆风顺。

在河南殷墟出土的甲骨文上,有用朱砂和墨书写文字的痕迹,表明在甲骨文上书写的文字,红色是朱砂,墨色是碳素单质,这证明朱砂和墨在殷代就开始被巫人用来书写文字了。在商代石、玉、陶器的表面,也曾发现过墨书的遗存。墨的起源较笔为早。起先人们将墨与朱砂都作为一种绘画的颜料来使用,不过早期的墨都是采用天然材料,甚至用墨斗鱼腹中的墨汁为墨,进行绘画或染色。但是由于文字的广泛普及,墨的大规模生产就变得非常必要了。

秦汉及魏晋时期是墨史上一个重要的时期,当时就有石墨、油烟墨、松烟墨之分。其中,石墨即石油燃烧所制之墨;油烟墨则以燃油所获烟炱制做之墨;松烟墨则是燃烧松木所制之墨。最先得到规模发展的制墨方法是油烟墨,其制作方法并不复杂:找一个易燃的烛心,放在装满了油的锅里燃烧,锅上盖好铁盖或呈漏斗形的铁罩;等到铁盖或漏斗上布满烟炱,即可刮下来,集中到臼里,加入树胶,混合搅拌,使其成稠糊状;将

成稠糊状的墨团，用手捏制成一定的形状，或放到模具里，模压制成具有一定形状的墨锭。这是传统烟墨制法。随着文字使用的进一步扩大，松烟墨逐渐取代油烟墨成为最主要的制墨法：与油烟制墨法原理差不多，通过燃烧松木来获取松烟粉末，然后与丁香、麝香、干漆和胶加工制成。工艺虽然复杂一些，但原料易得，而且价格低廉，迅速成为主要的制模方法和墨源。松烟墨的大量流行及“韦诞制墨方”的成型，使中国古代制墨工艺也在经历变革之后而进入了成熟期。

2.3.2 隶变的方式

笔与墨两种书写工具的成熟，使快速、简便和低成本的书写成为可能，为文字的变革提供了可靠的物质保证。

下面来谈谈“隶变”。首先要弄清楚什么是“隶”，“隶”即徒隶、官吏。笔者认为即使在最初也至少包含两大部门的官员，一部分是行政司法体系的公务员，也就是书隶，另外需要补充的是军队中的军事将领。秦朝崇尚法制，刑讼文案众多，需要撰写大量文书，而当时的官文篆书圆转回环，这就大大加重了公务员的工作量，而对于战事频繁发生的军队而言，篆书简直就是灾难。文字的书写效率与文字用途的矛盾就凸显出来了。这属于文字的第二类矛盾——文字与书写者之间的矛盾。

单从字体变化的角度而言，总的来讲，隶变就是将字形由圆形变为方形，笔势由纵势变为横势，线条由弧线变为直线，笔画由繁复变为简省。有关隶变的细节，在《试论汉字的隶变》一文中有详尽的说明。为了内容的完整性，现摘要若干重要的特点呈现如下，有关实例可以参考 2.2 节的对照实例。

1. 隶变中的“物理变化”

所谓物理变化，就是在变化过程中没有新事物生成的变化。在汉字隶变的方式中，有相当一部分汉字只是在之前文字的基础上对笔画、形体的厘定，至于表音、表义则与之前文字差别不大，我们姑且把这种变化称为隶变中的“物理变化”。

如表 2.1 所示，分别介绍了变曲为直、变直为曲、变断为连、变连为断、变繁为简、变长为短、点画位移 7 种基本隶变方式。在隶变过程中，这几种基本方式往往是交替综合使用的。例如，由繁为简就是变连为断、变长为短、点画位移三种基本方式的综合运用。

表 2.1 “隶变”对字形的影响

1. 变曲为直	变曲为直是指在隶变过程中,把原字中的曲线、弧线拉成直线或折线。直线的使用,使原字笔画变短,便于书写,字体更加疏朗	<p>𣎵(简石)——𣎵(简金)</p> <p>𠂔(石)——𠂔(金)</p> <p>𠂔(金陶)——𠂔(简)</p> <p>𠂔(陶)——𠂔(简)</p> <p>𠂔(石金)——𠂔(简金)</p> <p>𠂔(陶)——𠂔(简石陶印)</p> <p>𠂔(印)——𠂔(简)</p> <p>𠂔(石印)——𠂔(简)</p>
2. 变直为曲	变直为曲是指在隶变过程中,将原字笔画中的一些直线扭曲,以达到字势平稳、视觉效果美观的效果	<p>𠂔(简)——𠂔(汉简)</p> <p>𠂔(简)——𠂔(简)</p> <p>𠂔(简)——𠂔(碑)</p> <p>𠂔(刻石)——𠂔(碑)</p> <p>𠂔(简)——𠂔(简)</p>
3. 变断为连	变断为连包括两层意思:一层是隶变过程中发生的连笔现象;另一层是将原字中的某些线条延长,使之与另一笔画连为一体。变断为连使得字体结构更加紧凑、美观	<p>𠂔(石陶印)——𠂔(简陶)</p> <p>𠂔(金)——𠂔(石)</p> <p>𠂔(陶)——𠂔(石)</p> <p>𠂔(简印)——𠂔(碑)</p> <p>𠂔(简)——𠂔(碑)</p> <p>𠂔(简)——𠂔(碑)</p> <p>𠂔(简)——𠂔(简)</p>
4. 变连为断	变连为断是指隶变过程中,将原字中本来连着的字体断开,使字体结构更加疏朗,更合乎运笔的自然习惯,书写起来更有节奏感	<p>𠂔(简)——𠂔(碑)</p> <p>𠂔(简)——𠂔(碑)</p> <p>𠂔(简)——𠂔(简)</p> <p>𠂔(简)——𠂔(碑)</p> <p>𠂔(石陶)——𠂔(简)</p>
5. 变繁为简	变繁为简是指在隶变过程中删减去原字中的一部分笔画,使字体趋向简洁	<p>𠂔(陶)——𠂔(简)</p> <p>𠂔(陶)——𠂔(简)</p> <p>𠂔(简)——𠂔(简)</p> <p>𠂔(简)——𠂔(简)</p>
6. 变长为短	变长为短是指在隶变过程中把原字中的长线条变短,以调整简化字体	<p>𠂔(石陶)——𠂔(简金)</p> <p>𠂔(印兵陶)——𠂔(简)</p> <p>𠂔(石陶)——𠂔(简)</p>
7. 点画位移	点画位移是指在隶变过程中把原字的整体或一些部件做出适当的位移,以使字体平稳书写便利	<p>𠂔(简金)——𠂔(简)——𠂔(碑)</p> <p>𠂔(简)——𠂔(简)</p> <p>𠂔(简)——𠂔(碑)</p> <p>𠂔(简)——𠂔(陶)</p> <p>𠂔(陶)——𠂔(简)</p>

2. 隶变中的“化学变化”

所谓化学变化，就是在变化过程中有新事物生成的变化。在汉字隶变的方式中，也有一部分汉字是通过隶变使它所代表的音义发生了相应的变化，为区别于上文介绍的单纯字形上的变化，姑且把这种变化称为隶变中的化学变化。

1) 省形

如果省掉字中的某个形符，往往会导致固有结构的无理化。这类省形，属于“物理变化”中的“变繁为简”一类。但有些字，即使省掉了一部分形符，仍然不影响表音表义。如霍作霍（居甲 796A），省掉了一个“隹”，仍可以表示鸟在雨中飞，“其声霍然”的意思。

2) 省声

省声是指省掉形声字声符的一部分。按照汉字固有的形声原理，省略声符会影响该字的表音功能，但是有一类形声字的声符本身是形声字，省声省掉的只是这个形声字声符中的形符或部分形符，原有的声符保留了下来。这种形声字声符虽然省掉了一部分，仍然具有表音功能。譬如栖（睡 50·69），从木否声，把“否”声省作“不”声。栖虽然省掉了原声符的一部分，但仍可以继续表音。

3) 换形

换形是指在隶变过程中，一些字的形旁被意义相近或与原字相关的形旁替代。被替代的有的是象形或会意字，有的是形声字。如谿字（孙愍 109）从谷奚声，隶变后用“水”代“谷”，因为“水”与溪的字义有关。

4) 换声

隶变当中，有时会出现调换形声字声符的现象。如糧（银子 59）作粮（礼器碑），把“量”声换成“良”声，掉换了声符，但对表音、表义影响不大。

5) 增形

增形是指隶变中在原来象形表意字或形声字的基础上增加表义的形旁。如然（睡 23·12）是形声字，《武梁祠画像题字》作燃，加“火”旁。莫（睡 20·185）原为会意字，《彭庐买地券》作暮，加“日”旁。增形后，增强了表意功能。

6) 义符讹变为声符

在隶变中有些字，其象形表意的形体发生了变化，讹变为声符。如耽（马春 75）本

从心耳声，隶变后作耻，因为“心”与“止”形相近，于是写字人把意符“心”改成“止”，而“止”成为新字的声符。

隶变之前，以往的篆字都是比物赋形用以表义，有很强的图画意味。这个时期的文字很不成熟，书写上具有很大的随意性。因此，不同地域对同一个字的写法不尽相同，导致读者的理解千差万别。通过隶变，原来的图画文字转化为抽象线条构成的符号文字，笔形也从圆转回环变成由点和直线构成的笔画。汉字增强的符号化特性，使文字的结构变得清晰、简洁、规范，不同地区的人阅读不会产生歧义。这是汉字逐渐走向成熟的一个重要的里程碑。

隶变不仅是古汉字一次重要的变革，也是古文字向今文字发展演变的重要转折点，是古今汉字的一个重要的分水岭。这次变革对后世的影响深远，后世的楷书、行书都是从隶书发展而来的。纵观中国五千年的文明史，字体的演变虽然缓慢、渐进，但却从未停止过。虽然隶书出现并很快通行起来，但与旧字体仍旧并存很长一段时间。到了汉、魏两代，小篆已经不通行了，而《说文解字》仍用篆书写成，但汉字的总体趋势是一个由繁难变简易的过程，这个趋势从未改变过。

2.3.3 汉字的符号化与结构

那么，什么是汉字的符号化呢？为什么汉字的符号化是汉字走向成熟的标志呢？

前面讲过，早期的文字，也就是甲骨文和小篆阶段的汉字，文字的最小划分单位为象形字，也称为独体字。在此之上通过指事、会意、形声、转注等造字法来构造出新的汉字，称为合体字。这样从结构上，语句中的单个字就包含了两种形式：一种是独体字；另一种是由独体字复合而成的合体字。

当一个独立的结构中包含着另一个完整的子结构，而这个子结构又可以作为另一种独立结构与其父结构同级使用时，这两种结构在识别上很容易产生混淆。在相当长时期的古文行文中，句子的划分没有统一的标点符号，有时候不容易划分出独体字和合体字。这样的文字在组成结构上的不一致给文字的识别带来了困难。汉字符号化的重要目的之一就是要解决这个问题。其解决的方法大致分为如下两个步骤。

第一，将原有边缘参差的图形化汉字——象形字，变为外形一致、大小相等的方块字。隶书阶段形成的字形是扁方形，到了楷书阶段就完全规整为正方形了。在书写时，每个字的大小都尽量相等，便于识别。

第二，对合体字和独体字进行各种形式的变形，使其内部的子结构失去独立性。也就是说，无论是合体字还是独体字，其内部的子结构都尽可能不使用独体字。如果无法避免，例如形声字的声部，那么这个独体字也要做变形处理。

通过一系列这样的措施，最终导致文字结构划分的最小单位产生了变化。汉字新生了一种统一的子结构，就是我们常说的部首，现代的文字研究中统称为部件。最早关于“部首”的记载出现在东汉许慎的《说文解字》中（以下简称《说文》）。在《说文》中，许慎以基本汉字“部件”为线索，把 9 353 字归并为 540 部，每部的首个字符即部首。部首是对形态相似的汉字的一种归类，属于从转注法角度观察和统计汉字的一种结果。每部之首因其统帅全部，具有一定的代表性，后来就逐渐发展成了汉字的部件，就是现代常称的偏旁部首。虽然《说文》一书并未出现“部首”一词，但后来研究学者都约定俗成地简称每部的首字为“部首”。这是汉字最早部件化的开始。

“偏旁”也是人们分析汉字结构时所使用的一个概念，它出现得要晚一些。其原因是从隋唐开始，官方逐渐废止了隶书，而开始通行楷书。楷书取代隶书是造纸术和科举制度的共同产物。一方面，纸的发明进一步降低了书写的成本，几乎中等以上的普通家庭都能负担得起读书、写字的费用。另一方面，科举制度使每个人都有机会通过学习文化而改变命运。因为考生的数量逐年增多，为了降低阅卷的成本，就需要通行一种更加简洁、美观大方的标准化字体，要求全国考生统一学习，作为答卷的统一用字。这是楷书得以兴起的内在原因。因此，文化和教育再一次得到了空前的发展。笔者认为，楷书的通行加速了汉字符号化的进程，而文字研究在“隶变”之后又一次达到了新的高潮。

对楷体的研究，再次涉及文字符号化的问题，“偏旁”一词作为专门术语，从唐宋时期开始在学者间通行起来。“偏旁”最先应该特指形声字类型合体字的左右两部。所谓“偏”是指左右结构合体字的左方，那么“旁”就是指合体字的右方。白话文常以二字构词，就把合体字各部位的各个部件统称为偏旁。

到了现代，偏旁和部首逐渐合成了一个词，它们都是指汉字的零件，比较正式的说法叫作“部件”。也就是说，汉字本身并不是构成书写的最小单位，偏旁部首才是汉字构成的最小单位。这么做是比较科学的，因为汉字数量庞大，特别在文言文中独字成词的现象很普遍，常用字就有 7 000 多个（现代汉语降为 3 000 多个），汉字总数近 5 万个（《康熙字典》）。即便历经多年寒窗苦读，单独记忆这么多字也不是一件容易的事情，更何况还要了解其读音和含义。

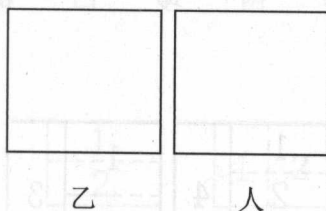
如果把汉字构造成偏旁部首库中某些元素的某种排列组合，也就是说，不论多么生僻的汉字都进行偏旁部首化分解（符号化）处理，将其限制在数量少得多的部件组合范

围之内,那么汉字的构造就变得有规律得多。这样,任何一个汉字的构成就可以分解为如下两个步骤。

- 一个字所包含的偏旁部首。有关所有偏旁部首的列表,现在基本上可以从任何一本汉语词典中都能查到,有兴趣的读者可以从如下网址中找到: <http://baike.baidu.com/view/1433394.htm>。
- 这几个偏旁部首以何种方式布局。也就是说,这些偏旁部首在位置上的排列组合。

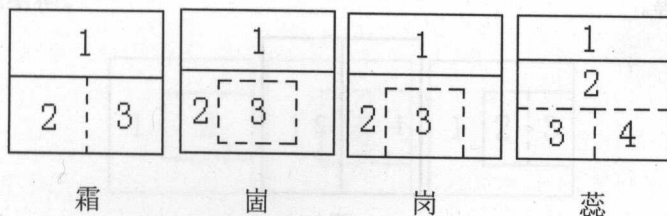
根据汉字部件之间的位置关系,汉字结构的基本类型可以分为若干种,一说7种,一说8种。这里借用二级子结构的方式来说明,尽可能地包含所有的结构类型。将最初形成的独体字结构作为一类,合体字的每一类分为若干变式。下面列举一些常见的例子仅供参考。

1) 简洁而不可再分的独体字结构: 字占格子中央, 方方正正

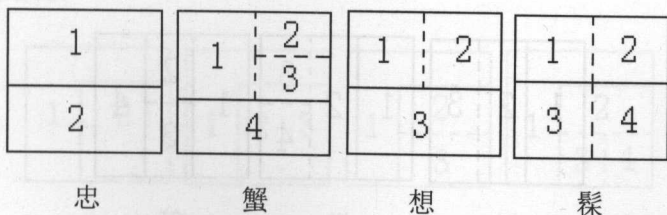


2) 上下结构

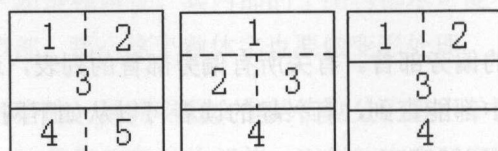
(1) 上小下大。



(2) 上大下小。



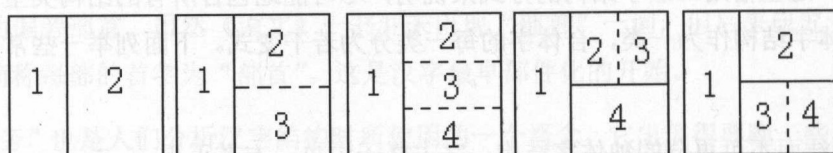
(3) 上中下结构。



器 孽 翼

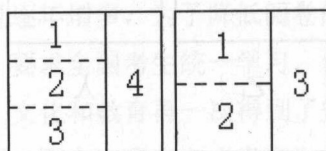
3) 左右结构

(1) 左小右大。



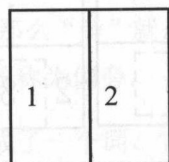
村 楼 塍 撵 撬

(2) 左大右小。



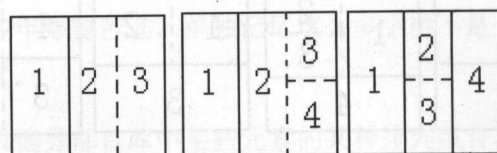
剽 剽

(3) 左右相等。



朋

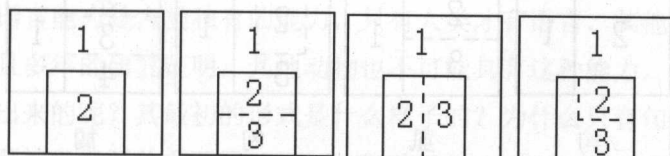
(4) 左中右结构。



锄 搬 掰

4) 半包围结构

(1) 上三包结构。



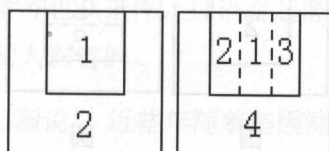
风

周

网

冏

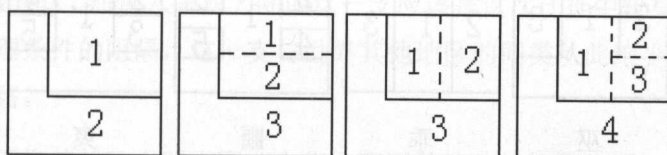
(2) 下三包结构。



凶

函

(3) 左下包结构。



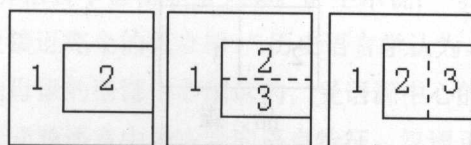
廷

迢

迦

邂

(4) 左三包结构。

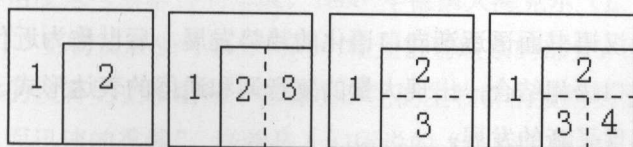


区

匿

匱

(5) 左上包结构。



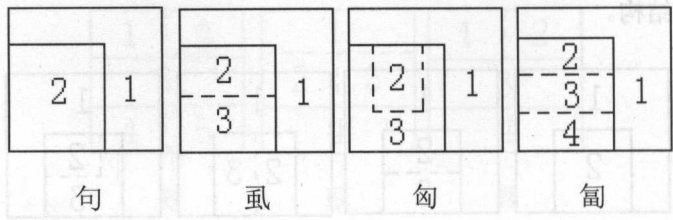
庆

屁

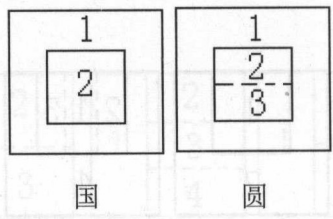
屢

屨

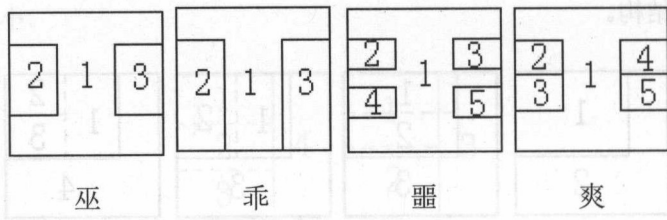
(6) 右上包围结构。



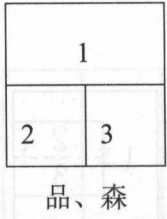
5) 全包围结构



6) 对称结构 (或称框架结构)



7) 品字结构



我们在记忆汉字时，首先单独记忆数量小得多的偏旁部首，再根据各个偏旁部首的位置和结构来记忆各种生僻的汉字，就会使汉字的学习变得容易多了，从而极大地降低了汉字的学习难度，即便现在看来也是一种极为精妙的思维！

在隋唐之后，汉语书面语逐渐向口语化的趋势发展，后世称为近代白话。它的一个鲜明的特征就是与口语相结合，出现大量的复音词和通俗的表达形式。此时，以独字为核心的古汉语又出现了新的发展。

2.4 汉语的发展

自然界中,语言能力是人类独有的能力,只有人类才有语言,其他任何动物都不具备这种能力,而且多年的研究证明,其他动物也不可能具有这种能力。那么,语言是从何时被人类创造出来的呢?其最初的形式是什么样子的?为什么只有句子才能完整地表达语义?这些问题就显得至关重要了。

关于人类何时形成语言这个问题已经研究讨论了很久,学术界一直没有定论,在1866年,巴黎语言学会甚至发表通告,禁止在学术会议上讨论该问题,有关期刊也不再发表此类论文。即便如此,科学界也从未停止过语言起源的研究,虽然迄今为止仍未有定论,但一些重要的成果还是发人深省的。

首先是人类的“非洲单一起源说”。近些年随着基因测序技术迅猛发展,古人类学家已经证明,大约在10万年前,现代人类的祖先全部生活在非洲东部,5万~6万年前,现代人类从非洲逐渐发展起来,向世界各地分期分批地迁徙,有的就近进入欧洲大陆,有的通过欧洲大陆到达亚洲,有的则去了美洲大陆。其中某一支在途中与他们的其他同类分手,来到东亚地区,而进入东亚大陆的这一支应该是这次迁徙中较早的一批。此后,也许因为特殊地理条件的阻隔,这一支与世界其他地区的同类从此绝少交流,走上了完全不同的发展道路。

众所周知,不像其他的文明古国,中华文明因其特殊的地理位置,幸运地得以一脉相承,汉语也幸运地、较多地保留了象形特征。而其他部族的语言,则由于交流的需要,都以记录语音符号为基本要素,显得与汉语大相径庭。因此,今天的汉语(也可扩大为汉藏语系诸语言),在世界语言中显得非常另类。萨丕尔说:“我们发现汉语比我们可能找到的任何其他例子都更接近完全的孤立语。”历史语言学认为,从同一语言分化出来的各个语言,其中离开原始母语的语源中心越远的,受语源中心的变化影响就越小,因而可以在这种语言中找到同源语言中最古老的语言特征。汉语正是如此。读完本节的内容读者就会了解到,由于地域特殊导致了汉语的保守性,在这4000年中,汉语基本的语法格局始终没有本质的变化。

现在从另一个角度来考察语言的形成。1866年德国人海克尔(E. Haeckel)在《普通形态学》中提出,“生物发展史可以分为两个相互密切联系的部分,即个体发育和系统发展,也就是个体的发育历史和由同一起源所产生的生物群的发展历史,个体发育史是系统发展史的简单而迅速的重演”。这就是人们所说的“生物重演律”。

儿童语言获得过程可以看作人类语言发展过程的浓缩性重演。Moskowitz 经过多年对儿童语言习得的研究和观察，将儿童语言称为“电报式言语”。他发现，“在儿童语言的第一阶段，其句子最长只有一个词；其后的阶段最长句为两个词。”注意，这里最长句为两个词，但没有三词句阶段。而这两个词一般都是什么词呢？他又说，“最初学会的词汇基本上都是具体名词和动词；尔后才是比较抽象的诸如形容词之类的词”。他描述这种类似电报式的语言为：“许多基本的语义关系都是由两个单位表达的。”“早期电报式的语言的特征是句子简短，基本上是由实义词构成的简单句，这些词有丰富的语义内容，通常是名词和动词。这种语言之所以叫作电报式的语言，是因为这些句子中没有功能‘词’，即没有动词时态词尾，没有名词复数词尾，也没有前置词、连词、冠词，等等。”

如果我们从信号的角度来看待这个问题，答案就会很明朗。人类在单词句阶段的语言，严格意义上不能称为“语言”，它更像“动物的叫唤”，动物也会通过嚎叫来发出信号，或者警示危险、或者宣誓主权、或者发现食物、或者表达情绪，这并不奇怪。而一旦进入“双词句阶段”，相当于句子最初的形态，称为“指称—陈述”的分化阶段，此时最初的语言就萌芽了。“指称—陈述”的分化，意味着名词、动词、语法三者也逐步分化出来，语言就自然而然地诞生了。

2.4.1 完整语义的基本形式——句子

如果上述对语言产生之初的状况的推定是正确的，那么需要回答的仅剩最后一个问题，为什么只有句子才能完整地表达语义？

应该讲，对于自然语言处理的研究，回答这个问题比前两个问题更有意义。前两个问题是在回顾历史，是历史问题或者考古问题，而最后这个问题却迫使我们不能面对现实。

语言的本质是一种信息编码。那么，信息的范畴又包括什么呢？相对于语言而言信息的范围更广泛，它指音讯、消息、通信系统传输和处理的对象，泛指人类社会传播的一切内容（来自百度百科）。这个定义有点空泛，不容易被理解。信息论的奠基人香农（Shannon）认为，“信息是用来消除不确定性的东西。”这一定义后来常被人们看作有关信息的经典定义加以引用，但笔者认为仍旧不清晰。为了更好地理解信息的本质，我们考察一下有关世界本质的如下描述。

直觉上，我们生活的世界是由空间和时间构成的，但从本质上讲，世间万物是由物质和能量构成的。引用一个流行多年的观点：世界由物质（这里应该称为质量）、能量、

信息三大要素组成。说实话笔者一开始不完全理解这个观点。首先,有关物理学的研究中,物质和能量是两个经典的问题,而信息则不同,在这个观点中的信息好像很突兀地被加了进来。而且,爱因斯坦早就说过:“质量就是能量,能量就是质量。时间就是空间,空间就是时间。”有关质量和能量的转换方程,爱因斯坦的相对论早就给出过。而时间和空间的统一性理论,早已被斯蒂芬·霍金证实。那么,再回到这个问题:什么才是信息,信息的本质究竟是什么?

虽然讲到这里,我们还是搞不清楚,但是隐隐约约可以感觉到,已经越来越接近最终的答案了。我们不想引用大量的物理学原理和数学方程,而是从思维最深处来探究。既然质量和能量可以相互转化,也就是说质量和能量都可以代表同一事物,是同一事物在不同环境中的不同状态。那么,质量说明了物质的什么属性,而能量又说明了物质的什么属性呢?有一定的物理学和哲学方面常识的读者,应该不难回答这个问题:质量的本质是存在,而能量的本质是运动。一般来讲,纯粹的能量或纯粹的质量都不存在,世界的万事万物总是在存在和运动之间求得一个平衡。之所以称为存在,是因为有相对稳定的结构,结构创造了空间;之所以称为运动,是因为其轨迹经历了时间。所以,空间是存在的测度,时间是运动的度量。

简单地解释了客观世界万事万物的基本原理,接下来再看看信息。维纳说:“信息就是信息,既不是物质也不是能量。”这句话看似说了,又看似什么都没说,但再仔细想想,还是说了。以往我们对物质的研究都集中于物质本身,而信息则不同,信息的目标在于解决事物与环境的基本联系。为了便于说明,我们看看《信息论》中有关通信系统模型,如图 2.16 所示。

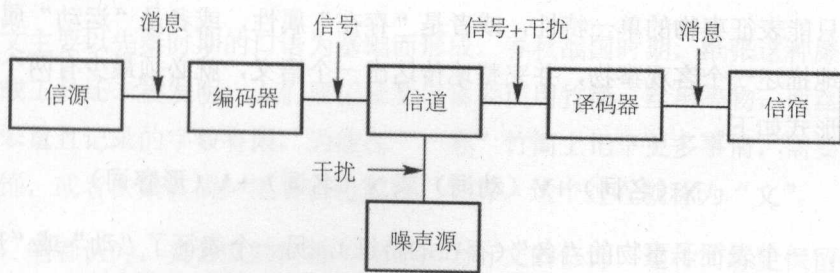


图 2.16 通信系统模型

通信系统主要分成如下 5 个部分。

- (1) 信源。顾名思义,信源是产生消息和消息序列的源。在自然语言中可以认为是外部世界的信号。

(2) 编码器。编码就是把消息变成适合在信道传输的物理量。这种物理量称为信号。可以理解成一种能够感知的存在，可以是能量形式（光信号、热信号、其他辐射信号），也可以是质量形式（大小、质量、速度等）。

(3) 信道。信道是指通信系统把载荷消息的信号从发送端送到接收端的媒介或通道，就是人们生活的自然环境和社会环境。

(4) 译码器。译码就是把信道输出的已迭加了干扰的编码信号进行反变换，变成信宿能够接受的消息。大脑通过感官接收到外部传递的视觉、听觉、触觉、嗅觉等信号，通过神经网络进行处理，最终变为可以理解的语句。

(5) 信宿。信宿是消息传送的对象，这里特指接受消息的人。

从上述不难看出，信息编码了外部对象的质量和能量，并通过解码使信宿本身能够感知存在。形象地说，信息是信宿对信源存在的感知。广义上说，它反映了一个事物在特定时空中对另一个事物的感知。因此，信息的本质是一种联系。失去了联系也就失去了信息。与质量和能量不同的是，信息联系了两个或两个以上的事物，其中一个事物使用某种编码表征了另一种事物。因此，我国著名的信息学专家钟义信教授曾对信息给出如下定义：

“信息是事物存在方式或运动状态，以及这种方式或状态直接或间接的表述”。简而言之，信息就是对事物存在和运动的表征。

存在和运动都是事物最本质的属性。一种编码系统，想要完整地表达任何一个客观的事物，必须既要有能力表达其存在特征，也要有能力表达其运动特征。语言也不例外。单个的词只能表征事物的单一特性，或者是“存在”属性，或者是“运动”属性。语言想要完整地描述一个客观事物，并完整地传达出一个语义，就必须最少有两个不同性质的实词，形式如下。

N（名词）+V（动词）或 N（名词）+A（形容词）

其中，一个表征了事物的“名”（存在的特征），另一个表征了“动”或“形”（运动或属性的特征）。这自然而然地导致早期人类语言的用词在词性上的分化。这种分化也称为“指称—陈述”分化。只有实现了“指称—陈述”分化，语言才能称为语言，而不是野兽的“嚎叫”，人才能称为“人”，而不是什么高级动物。

人类最初的语句带有很强的原始“指称—陈述”分化的痕迹，其表现形式多为 SV（主、谓）结构。这一点也不难理解，刚刚学会使用语言的人类，就如同刚出生的婴儿，

描述客观世界的方式,必然是简单、幼稚的。根据基本的思维形式和逻辑形式,可以认为,人类最初的语言是一种只有本体名词和相应的实义动词的语言,其他语法成分都是后起的。然而,早期语言的这种分化并不彻底,在汉语中,除去虚词和某些特殊用途的词之外,很多常用词的词性迄今为止仍没有固定下来,而需要根据上下文来确定。

但是,随着时代的变迁,语言的发展也越来越成熟,句子结构也变得越来越多样,终于形成现代语言中最常出现的SVO(主、谓、宾)型的句法模式。同时,随着语言成熟程度的提高,“指称—陈述”分化也越来越彻底。现代西方的语言,特别是英语,名词、动词、形容词之间的分化已经完全可以通过词汇的形态变化加以区分。

2.4.2 语言的初始形态与文言文

人类的语言形式、思维形式、逻辑形式最初本是三位一体的,也就是我们常说的“言由心生”。表达形式(指称—陈述)、思维形式(本体—属性)和逻辑形式(主词—谓词)的高度一致性也深刻地体现在汉语中。特别是在古汉语中,一切的词汇和句式都是为了清晰、准确、生动地传达语义而存在和变化的。常以表现强烈的画面感、内在情绪等为主要目标。

因为历史久远,古代汉语是较为接近人类语言初期状态的语言。著名语言文学家、教育家王力先生在《古代汉语》中指出:“文言是指以先秦口语为基础而形成的上古汉语书面语言。”使用古代汉语所写成的文章称为文言文。第一个“文”字,为“纹”,修饰之意。中间的“言”字,是口头所说的话,也就是口语。“文言”两字,就是修饰过的口头语言。最后一个“文”字,是作品、文章等意思,表示文种。

文言文主要以先秦时期的口语为基础而形成。春秋战国时期,纸张这种廉价而轻便的文字记载工具还未被发明,人们要记录文字需要使用竹简、丝帛等物,而丝帛价格昂贵、竹简笨重且记录的字数有限,为能在“一卷”竹简上记下更多事情,需要对语言做一定的修饰,或者去繁就简,或者去粗取精,等等。这个过程就称为“文”。

不过,笔者认为,文言文的简练不完全因为行文的修饰,这种简练更保留了人类语言初期的形式和特点,即表达形式(指称—陈述)、思维形式(本体—属性)和逻辑形式(主词—谓词)的高度一致性,这一点对理解古代汉语非常重要,与相对较为成熟的现代汉语相比,其主要表现在如下几个方面。

(1) 词法上强调本体和属性,陈述中为了明确语义常以本体或属性充当谓词。

□ 名词用作动词、使动词、意动词,或者直接作状语。“一狼洞其中”中的“洞”

表示山洞——名词性的本体，而这里为了明确语义作为动词使用。“纵江东父老怜而王我”中的“王”为“使……称王”的意思，属于使动用法。

- 形容词用作名词、动词或者使动词。例如，“将军身披坚执锐”中的“坚”指“坚硬的铠甲”，“锐”指“锐利的兵器”，是形容词用作名词。再如名诗句“春风又绿江南岸”中的“绿”字，则是典型的形容词使动用法，意为“使……变绿”。这是属性做谓词的鲜明示例。
- 数词用作动词。如“夫金鼓旌旗者，所以一人之耳目也”中的“一”是“使……一致起来”的意思。
- 动词用作名词、使动词。如“子钓而不纲，戈不射宿”中的“宿”指“歇宿的鸟”，动词当作名词。

在现代文中，上述这些词的词性基本都已经固定下来了，而在语言的形成初期却不固定。应该讲，当时人们很难找到合适的动词来表达如此准确、生动的语义，其实现在也难。虽然现代的语法称之为词类活用，但从认知角度来看，这些词的活用是为了更好地表现语义，使语义变得更具体、生动。由此可见，古代汉语在词法的应用上充分体现了重语义、轻文法的特点。这是汉语（包括古今）一直以来非常重要的一个特征。

（2）句法上的逻辑性。

- 语序上轻文法、重语义。现代汉语句式一般是“主语+状语+谓语+宾语”的顺序，古代汉语中的语序则较为特殊，主语可以在谓语后，宾语可以在谓语前，状语可以在谓语后，等等。用现代句式套用文言句式寻找差异，实在没有什么意义。我们可以一言以蔽之，对于文言文，你强调什么就把什么放到重要的位置，或者放到句子的最前面，或者放到最后面，然后再组织其他的成分。

例如，“居庙堂之高”译为“处在高高的朝堂之上”，是不够准确的，古人的本意是“处于像朝堂那样的高处”，想强调的是“高”，而不是庙堂。再比如，“甚矣，汝之不惠”，我们常看作主语和谓语倒置；定语后置句，应为：“汝之不惠，甚矣。”（“你太不聪明了！”）其实古人的本意是“太不聪明了，你”，想强调的是“太不聪明”（类似现代文中的表达：见过不聪明的，没见过像你这样不聪明的）。有了前两个例子，后面这个句子就很容易理解了：“石之铿然有声者，所在皆是也。”“那些（敲打时）能铿锵作响的石头，到处都是。”这里想强调的是“石头”，而不是“铿锵作响”。

- 句法用于表示逻辑句式的不完整。

从逻辑角度讲，古文中的判断句仅是一种意念的判断，还未出现命题判断的谓词。

例如,“陈胜者,阳城人也。”现代汉语的判断句常以“是”作谓语。这句话在现代汉语中可以翻译为:陈胜是阳城地方的人。

被动句中,现代汉语中的被动句由介词“被”构成,而古代汉语中很少使用“被”显式地标示被动。语句中仅用动词表示,被称为意念被动。这种无标示的被动词需要在具体语境中加以理解,例如,“文王拘而演《周易》”中的“拘”是“被拘禁”的意思,属于被动句。

再来回顾那句话:人类最初的表达形式(指称—陈述)、思维形式(本体—属性)和逻辑形式(主词—谓词)是高度一致的。读者应该能够理解其中的含义了。汉语,特别是古汉语,它的基本语法形式与当时人们的基本思维形式和逻辑形式是高度一致的。

这也体现出文明正处于形成期,还有诸多不成熟之处。所以,在句式上强调判断和被动的句式并不完备。这不仅反映了语言的不足,也体现了人们思维的不足,特别是,逻辑思维尚处于萌芽期。汉语从此与西方语言重逻辑、重形式的道路分道扬镳,形成了一条特有的发展轨迹。在描述社会生活、自然事物、内心情绪上,古汉语已经达到了相当的高度。以至于几千年的封建王朝中,文言文成为唯一的官方文体。而在古汉语中,语义绝对是第一位的,为了服务于语义,可以灵活地调度文法、甚至不顾词性的差异。所以说,与西方的拼音文字不同,汉语的一切变化都是以表意性为核心的。

2.4.3 白话文与复音词

所谓“白话文”是指一种书面语,而不是将口头语言原封不动地直接复写为文字。它是根据常用的、直白的口头语言撰写的文章。这种文章中所使用的句法和词汇与口语基本一致,但也有修饰的成分。我们常说的“文言文”就是先秦时期的白话文,只是后世人们的语言发生了很大的变化,但仍使用先秦白话文作为官方书面语,于是称之为“文言文”。

语言像文字一样,随着时代而不断变迁。后世常指的白话文是由于口头语言逐渐发生了改变,与官方的书面语逐渐分离,而形成的一种独立文体。虽然文言文与白话文在句法上有一定差异,但总体都是一致的,所以我们并不觉得读文言文像读外语一样。造成文言文晦涩难懂的原因是几千年来汉语词汇的变化。从语言处理的角度而言,如果不去研究词汇沿革,中古和近代的白话文没有什么差别。

在介绍构词法之前,对词汇做一个简单的分类。根据词汇的生命周期,将词汇分为基本词汇和一般词汇。所谓基本词汇,是使用频率较高、生命周期较长、变化比较慢的

常用词汇，它们构成了语言系统的骨架，也是构成新词的基础。所谓一般词汇，是指那些随着新事物和新概念的产生和发展，而不断地产生的新词和新语；同时，又有一些旧词随着社会中某些事物和现象的消失而消亡，它们的生命周期比基本词汇要短得多。

再根据词汇的音节数划分，分为单音词和复音词。单音词，就是单音节词，由一个音节构成，在书面形式上用一个汉字表示，比如，天、人、牛、红、一、飞、走。文言文中多用单音节的词。复音词是由两个或两个以上的音节构成，就是由两个或两个以上的汉字构成的词，比如，风景、前途、人间、依附、脱落等。

词汇的音节数是区分文言文和白话文的一个重要依据。据统计，在现代汉语中，频率最高的1万个词中，单音节词占24%，其余均为多音节词。其中，双音节词占全部的63%。随着词汇统计量的扩大，双音节词占有的比例越来越高，而单音节词显著降低。文言文则一直以单音节词为主，占80%以上。

汉语的言文分离（口语与书面语的分离）从东汉末年就开始了。因为文言文在整个封建社会中都作为官方的书面用语，导致汉语的言文分离前后历经三次较大的变革。这三个时期包括：魏晋南北朝时期、宋元时期、鸦片战争到五四运动。历经近1700年的历史，直至发展到现在使用的白话文。

基本上，三个时期的共性都是相同的，即每个时期都伴随着外族的入侵，一方面对原有正统的中华文化产生了冲击，多种文化交织、融合使传统文化融入了新的元素；另一方面，农业、工商业和社会生活等全方位的变化也导致了语言的发展和进步。应该说这是汉语词汇变化的外因。

在漫长的汉语发展史中，词汇发展的规律与文字的发展规律截然相反，词汇从先秦那种简单，甚至简陋的独字词形式（很多证据表明，上古时期的独字词不完全都是单音节），向语义含义更加丰富的复音词，主要是双音词方向发展。词汇所表达的语义变得越来越细腻、精确和丰富了，词汇的总量也在不断增加，这里既包括基本词汇也包括一般词汇。例如，《世说新语》中有“风景、前途、人间、依附、脱落、克服、经营、修改、慨慷、正直、奇丽”等；《搜神记》中有“时期、时节、动作、从事、嫉妒、分别、努力、扶持、禁止、供养、怯弱”等。

词汇的复音化，特别是双音词的广泛使用有其内在的原因：随着经济的发展和文化的融合，新鲜事物不断涌现，人们需要描述的新事物越来越丰富，使得现有的词不够用，需要大量而频繁地构造新词。解决这个问题的策略如下。

第一，大量使用假借字，仍旧保持独字为词。这种方法的弊端前文已经讲过。大量

使用假借字容易造成假借字本身在语义上的混淆,不利于词汇准确、清晰地表达语义。而汉语是一种语义为先的语言,这么做的空间显然不大。

第二,使用造字法,根据新事物造出新的独字词。造字是一个复杂的过程,新字从制订、审核、发布,到最后通行是一个较长的过程,这个过程在现代都需要专门的部门投入大量人力、物力宣传推广,而在古代,大规模频繁地构造新字显然是不可行的。前文谈过有关基础词汇和一般词汇的概念。新事物的命名应归为一般词汇的范畴,大多数新生事物的生命周期都有限,如果每个事物都通过新字来表示,肯定是不现实的,这会造成大量的独字词还未推广被人们接受,该事物就已经消亡了。

第三,人类能够记忆并熟练使用的符号总量是有限的,迄今发现的几十万片的殷商甲骨中所使用的甲骨文符号也不过4000多个,其中有很多异体字。即便到了现代,虽然词汇的总量已经超过百万,但是所使用的常用字也只有三四千个。《康熙字典》所记载的超过4万多个汉字中,其中绝大多数都是生僻字,即使在当时文言文中也不常用,更不要说来源于老百姓口语的白话文了。所以,造新字这条路也行不通。

最后,只剩下一条路了,使用复音词。通过某种构词方式将若干个单音词复合使用,构成新词。这种方法并非新策略,在形声造字法中就已经被广泛使用。多数形声字仅分为左右两部,左边常是形部,右边是声部,两部合并构成新字。值得注意的是,汉字中形声字所占比例具有绝对多数,而复音词特别是双音词就借助了这种组合的原理。所不同的是,合成双音词的左右两字,是按照汉语句法规则进行搭配的,有如下几种类型:偏正式(属性+概念)、述宾式(动作+对象)、联合式(概念+概念)、主谓式(主体+动作)、述补式(动作+结果)等。将句法规则应用在造词上,是一个创举。这使得每个只要掌握口语的人都能准确理解复音词的含义,而事实也证明,这条路是成功的。

从魏晋南北朝开始,双音词的数量一直呈上升趋势,在之后的每个时期都出现了较大的发展,最终成为构成汉语词汇的主流。

非常有意思的是,在复音词的构造中,还呈现这样一种规律。汉语中双音词显著多于其他类的词汇,而且单音节和其他多音节词汇有不断向双音节变化的趋势。这种趋势不仅发生在古代,而且现代仍然如此。

(1) 单音节词汇变双音节的例子。

月—月亮

耳—耳朵

唇—嘴唇

发—头发

竹—竹子

石—石头

师—老师

姨—阿姨

(2) 多音节词汇变双音节的例子。

落花生—花生	照相机—相机	山茶花—茶花	机关枪—机枪
化学工业—化工	文学艺术—文艺	超级市场—超市	
彩色电视机—彩电	人民代表大会—人大	政治协商会议—政协	

这种趋势也反映出这样一个数学原理：假设按照人们能掌握的常用字 4 000 字来计算，双音词的数量就是 4 000 的平方，共计 1 600 万种组合方式。这个数字足够承载得下工作、生活中所需的所有词汇。而三音词、四音词在书写上明显比双音词要麻烦，至少没有双音词那么简练。因此，其他复音词向双音词转化就不难理解了。单音词成型的历史比较久远，一部分已经固化为固定的用法，扮演着不可取代的句法角色，这部分词的变化不大。一些常用的名词，为了语言韵律的一致性，也通过添加一个后缀或前缀演化为双音词。

汉语词汇走入复音节之后，词汇的表现发生了巨大的变化。

词汇所表现的内容更加丰富了，新词被大量创造出来，具有了时代的特征。

原有的独字词与其他词结合之后，语义变得更加丰富，用法也更加灵活，如“打”这个词，跟不同的词相结合，就有不同的意义。例如，“敲打、打围（打猎）、打迭（安排）、打水、打躬、打发、打点、打听、打探、打招呼”等（现代还有“打油”、“打伞”、“打盹”、“打枪”等）。

新词可以通过音译直接而得，例如，“逻辑（Logic）”、“引擎（Engine）”、“白兰地（Brandy）”、“沙发（Sofa）”、“扑克（Poker）”、“硼酸（Boricaacid）”、“摩托车（Motoracar）”、“冰淇淋（Iceacream）”、“东亚（Oaet Asia）”等。还有就是意译，如“银行”、“国会”、“电话”、“墨水”、“天使”等。

更多的是根据不同的时代创造出了具有本时代特征的新词。例如，“五四”期间出现了“实用主义、共产党、唯物主义、唯心主义、辩证法、经验主义、机会主义、关门主义”等新词。抗日战争时期又产生了“八路军、新四军、儿童团、决死队、地雷、民兵、伪军、沦陷区、汉奸、地下军、促进会、统一战线”等新词。建国后，还有“肃反”、“三反”、“五反”、“反右派”、“反右倾”、“四清”、“文革”、“四人帮”等词汇。这些词汇都有鲜明的时代特征。

很多一般词汇随着时代的变迁，逐渐退出了人们的视野，但是也有很多词汇沉淀下来，成为基础词汇，扩充了基础词汇的容量。

除音译词之外,复音词的新词构造与句法基本一致,它的理解和掌握都不需要外界的干预,也不需要特别的解释和说明。望文即能生义。这使人们自然而然地融入时代之中,既是新词的使用者,也是新词的创造者。

(1) 常用字的数量变少,降低了掌握文化的难度。

通过科举考试做官是古人学习文化的最主要的动因。古时科举要考的内容首推十三经,八股文是从十三经中随意抽取一个词、一句话、一段文字,就以此为题作文。所以十三经是必背的。十三经有多少字呢?据南宋郑畊老统计,周易 24 207 字,尚书 25 800 字(近人黄侃除去伪古文,则有 17 925 字),毛诗 39 224 字,周礼 45 806 字,仪礼 56 115 字,礼记 99 020 字,左传 196 845 字(孔子春秋本文 18 000 字),公羊传(清阎若璩统计) 44 075 字,谷梁传(清阎若璩统计) 41 512 字,论语 13 700 字,孝经 1 903 字,尔雅 13 113 字,孟子 34 685 字,大学 1 753 字,中庸 3 568 字,共计 641 326 字。其中,互不相同的单字数为 6 544 个字。

看起来不多,但由于文言文与现实生活严重脱节,又常用单音词,独字成词,词形与词义没有必然的联系,要写出像样的文章,不仅要记住这 6 000 多个字,而且对其使用语境,即所有十三经的 64 万多个字都要背熟才行。这绝不是古代科举故意难为人,现在的英语也是这样,英语中大多数单词都是独字成词,学习英语的语法并不困难,简单的会话也不是太大的问题,但是要用英语表达复杂的主题,恐怕不是一年半载就能做到的。

白话文则不同。山西大学计算机科学系利用计算机抽样统计从五四时期到当前的 200 万字的语料,检测选收常用字的使用频率。检测结果是:常用字(2 500 字)的覆盖率达 97.97%,次常用字(1 000 字)的覆盖率达 1.51%,合计(3 500 字)覆盖率达 99.48%,其他汉字只占 0.52%。现代常用字较之文言文时代减少了一半。原因很简单,有些能用复音词表示的语义,就没有必要再用单音词了。特别是体现语言丰度的名词类词汇。白话文的回归使文字不再是统治阶层的专宠,也不需要经年累月的艰苦学习,文字获得变得像语言一样简单容易,已经成为人们生活中必不可少的一部分。

(2) 产生了大量的同义词和近义词。

近义词和同义词的出现说明,词汇的总量大大超越了语义的总量。这有助于表现更准确和更细腻的语义,创作适用于各种语境下的文体。

例如,“看”这个动作,就有“瞧(细看、注视)、瞅(窥见、张望、远望)、瞄(shào 大概看)、包斜(要睡的样子看)”,还有“望、观、见、看”等。

下面几组都是常用的同义词和近义词:“照管、照应、照看、料理、招呼”;“留意、

留神、留心；注意、当心、小心”；“喜欢、欢喜、快乐、快活、开心、高兴”；“精壮、强壮、健壮、健旺”。

(3) 成语被当作一个词来使用。

在以复音词为中心的现代白话文产生之前，成语一般作为涉及典故的短句或词组使用。复音词产生之后，成语逐渐演变为一个词，具有固定和明确的语义，既使文章变得更加生动，也便于读者理解。

总之，随着词汇的发展，白话文已经为取代文言文做好了充分的准备。

2.4.4 白话文与句法研究

从语言学角度而言，上世纪轰轰烈烈的五四运动，也称为新文化运动，是语言和文字之间在表现形式上矛盾的集中体现，既是千百年来语言和文字严重脱节的一次革命，也是白话文文体的一次回归。前面讲过，语言和文字的两大矛盾，而这个矛盾是第一位的，是关乎到文字能否生存的本质问题。

胡适先生是这次运动的主导者之一，其在《文学改良刍议》中对新生的白话文提出如下著名的八大主张。

一曰：需言之有物

二曰：不模仿古人

三曰：需讲求文法

四曰：不做无病之呻吟

五曰：务去烂调套语

六曰：不用典

七曰：不讲对仗

八曰：不避俗字俗语

可以说，在白话文作为书面语言不到一百年的时间内，中国的语言学界基本上恪守这几条的原则，这使得白话文得以健康蓬勃的发展，并通过后来的简化汉字、汉语拼音、扫盲运动等一系列改革方案，使文盲率从解放前的 80% 下降到 2010 年的 9.04%，成人文盲率更少，仅为 4%。这个过程仅仅用了 60 年的时间。

在文化走进千家万户的时代，白话文的语法研究却没有那么幸运。真正意义上的汉语语法学，从建立到如今，也不过一百年的历史。19 世纪末至 20 世纪初，马建忠在模仿西方传统语法（拉丁语法）的基础上，撰写了第一部系统研究中国古汉语语法著作《马氏文通》。它的出版，标志着汉语语法学的正式诞生。最早汉语语法学基本上是借鉴西方传统语法学的框架和内容。五四运动刚一结束，20 世纪 20 年代，黎锦熙的《新著国语文法》则模仿英语语法建立起来，它是第一部系统的现代汉语语法著作，其出版标志着现代汉语语法学的正式建立。

到了 40 年代，吕叔湘的《中国文法要略》、王力的《中国现代语法》及高名凯的《汉语语法论》发掘并添加了汉语语法的一些特点，也有某些创新的观点，但整体上还是属于传统语法。随着对传统语法的研究和实践，人们发现根植于拼音文字的语法体系并不完全适合重语义、轻形态的汉语。

鲁川就将英语的语言结构比喻成“榫合法”，是把木料凿出“榫”（词汇的形态变化）而按照某种预定的框架（时态、语态）组合到一起。汉语是“黏合法”，无须木料变形，只把实词摆在一起，必要时用虚词黏合即可，并给出了一幅生动的示意图（见图 2.17）。

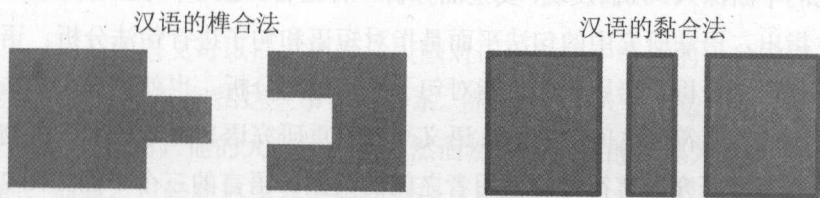


图 2.17 汉语与英语的结构比较

20 世纪 80 年代初的改革开放迎来一个新时期。新时期的学术政策宽松，人们思想开放、活跃，西方语言学的各种流派、各种学说都纷纷介绍到国内。其中，对汉语语法研究影响比较大的有转换生成语法、系统功能语法、“格”语法、从属关系语法、认知语法、话语篇章语法等。这些思想在本书的后面还有简单介绍，这里不再赘述。在对语法进行解释的问题上，国外的“形式主义”和“功能主义”在我国语法学界也得到了充分的研究。

现在语法学界有三派：一派主张从语言内部进行解释，如运用句法制约和语义制约来解释句子成立的条件或句式变换的条件，这主要受转换生成语法的影响（Stanford 的短语结构树和 PCFG 句法主要以此学派的理论为基础）；另一派主张从语言外部进行解释，如或用认知心理来解释某些语法现象、语法规则，或用交际功能来解释某些语法现象、语法规则，这主要受功能主义语法学的影响（本书的大部分观点都是基于这个学派的）；再有一派认为：有些语法现象只能从内部进行解释，有些语法现象只能从外部进行解释，

有些既涉及内部也涉及外部，由此主张在对语法现象或语法规则进行解释时，要具体情况具体分析，既要重视内部解释，也要重视外部解释。（这一派的成果似乎在前两派的思想 and 成果上做了一些集成。）

在此基础之上，语言学界对汉语语法做了如下新的诠释。

(1) 不依赖于严格意义的形态变化，而借助于语序、虚词等其他语法手段来标识语法关系和语法意义。

(2) 给出语法研究和学习的最终目标：揭示语义的决定性、句法的强制性、语用的选定性及认知的解释性。

2.5 三个平面中的语义研究

早在 1938 年，莫里斯提出符号学有三个分支：语形学、语义学和语用学。后来随着对语言研究的不断深入，人们发现，要全面分析一种语言，这三个方面都是必不可少的。胡裕树先生指出，语法研究中的句法平面是指对短语和句子进行句法分析。语义平面是指对句子进行语义分析。语用平面是指对句子进行语用分析。也就是说，句法平面着重研究不同层级的语言符号之间的关系；语义平面着重研究语言符号与所指事物之间的关系；语用平面着重研究语言符号与使用者之间的关系。语言的三个平面如图 2.18 所示。

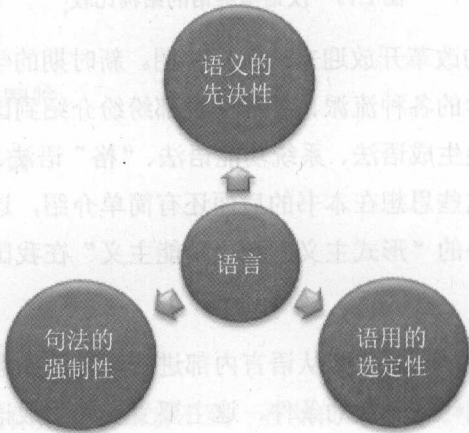


图 2.18 语言的三个平面

本节所说的三个平面与本书主体思想所论述的三个平面（第 4 章）大体一致，但在某些细节上存在差异。三个平面中的“语用”元素在本书的后续章节被更改为“语境”

(Context)。另外,关于语法和语境元素对中文,特别是对自然语言处理方面的影响,本书也做了修正。本书的第4章详细探讨了语法与语境对NLP算法设计和应用系统的影响。

因此,本节的内容仅仅涉及语义研究方面的成果,对于其他内容暂不涉及。

2.5.1 词汇与本体论

三个平面最核心的部分就是语义,对于汉语而言,语义对句子的生成和理解都具有先决的作用,影响最大。

所谓语义(Semantic)就是指信息(数据)的意义,放到通信领域,可以理解为编码的意义,具体到自然语言中,就是词的意义和句子的意义。词是人创造的,是后天、人为的。符号化以后的汉字失去了原来的象形特征,可以没有任何意义,现代汉语中占主要比例的词都是复音词(两个字或两个字以上)。这些词的含义一般都是在长期使用中,约定俗成而构成这样或那样的含义。

什么是意义本身呢?为什么会形成这些意义,或者说意义又是如何在人的大脑中组织起来的呢?要想回答这些问题,研究语义本身就显得至关重要。

直觉上,词汇的语义可以简单地看作人脑对现实世界中事物的一种编码——词汇。当然除事物本身之外,还包括这些事物的联系。例如,一个人去河边钓鱼,过了一会儿,钓到一条小鲫鱼。此时,他的大脑中就会自然而然地转换鱼的形象为词汇“鲫鱼”。这是显而易见的,而且,已经被科学界证明并复现(用机器模拟实现人类的视觉认知机能),也就是著名的卷积神经网络。看到这里,应该认识到第一个问题不难回答。

下面继续讲述这个例子。这个人很高兴地把钓到的鱼放到了鱼篓中,然后继续下钩。过了一段时间,又钓到了一条大鲤鱼,按照之前的原理,他的头脑中继续映射出词汇“鲤鱼”。之后他结束了一天的工作,回到家里。此时,他会跟妻子说:“今天运气真好,我钓到了两条鱼。”但是,为什么不说“今天运气真好,我钓到了一条小鱼和一条大鱼”或者“今天运气真好,我钓到了一条小鲫鱼和一条大鲤鱼”呢?(当然也不是绝对的。)

这反映了人类在组织概念上的一种重要的认知机制——范畴化。语义具有范畴化(领域性)的特征,不属于任何范畴的语义是不存在的。客观事物在人的头脑中形成的信息不是相互独立的,而是普遍联系的,随着对客观世界认识的不断丰富,逐渐形成一个网络。

在这个网络中,经常出现的关系有:表示概念的集合称为类,如上例中“鱼”就是个类。表示概念的元素称为个体,小鲫鱼和大鲤鱼分别是鱼类的两个个体。由小鲫鱼还

可以衍生出“鱼”类的子类：“鲫鱼”类；同样，大鲤鱼也可以衍生出子类：“鲤鱼”类。它们都属于“鱼”的子类，与鱼是上下位关系，在计算机行业也称为继承关系。

鱼身体呈纺锤形，表皮有鳞片，靠鱼鳍在水中游动，等等。这些特征都称为“鱼”类的属性和行为；而鱼没有肺，靠鳃呼吸，只能生活在水中，不能到陆地生活，这就是“鱼”类的约束。“鲫鱼”类和“鲤鱼”类也都继承了这些属性、行为和约束。

简单地说，当我们要定义一个概念（这里暂指类）的时候，要考虑如下几种特征。

- ❑ 类名：鱼。
- ❑ 子类名称：鲫鱼、鲤鱼等。
- ❑ 属性：身体呈纺锤形，表皮有鳞片，靠鱼鳍在水中游动。
- ❑ 约束：只能生活在水中，不能到陆地生活。
- ❑ 行为：在水中游动。

这就是最简单的“鱼”的概念，或者在语义领域以更专业的词汇来定义——本体。

本体论原来是一个哲学概念，古已有之。它用来研究客观事物存在的本质和组成。本体论希望对世界上任何领域内的真实存在做出一个客观的描述，而不依赖于任何特定的语言。近十多年的研究使本体超越了哲学的范畴，在与语义紧密相关的知识工程中，本体发挥了重要的作用。1998 年，蒂姆·伯纳斯·李（Tim Berners-Lee）提出语义网（Semantic Web）的概念，本体模型成为其中的核心概念，除此之外还形成了本体推理计算的基本规则。

本节主要参考鲁川对语义平面的研究结果给出一个本体的构建和计算的实例。

例句：“动物园里有许多多的鸟。”

构建意合网络（该数据结构在后面章节中更换为语义角色框架），如图 2.19 所示。

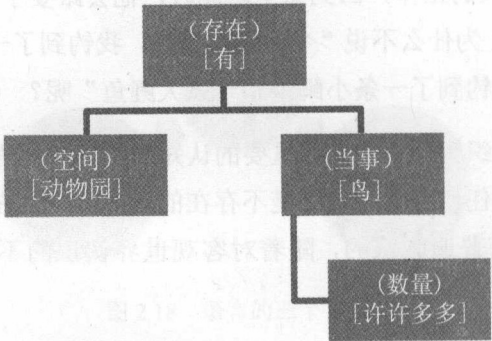


图 2.19 语义角色框架

激活“鸟”的本体框架，查看其上、下位结构，如图 2.20 所示。

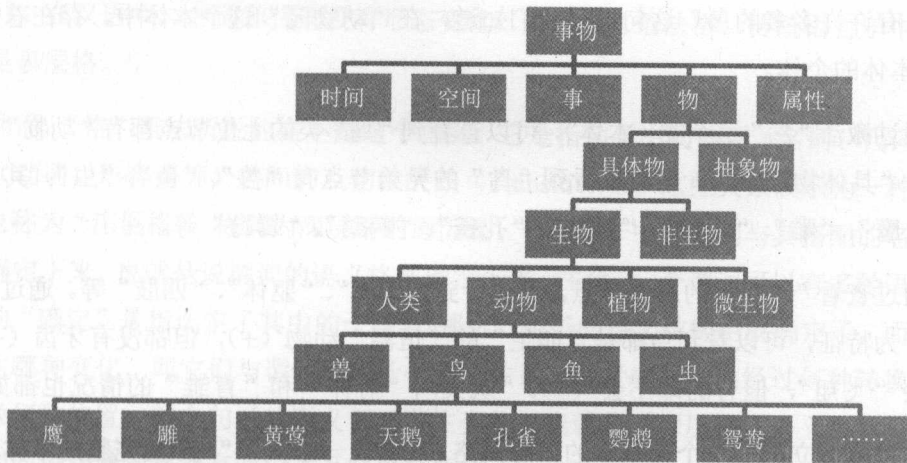


图 2.20 “鸟”的本体框架及其上、下位图

激活“鸟”的本体框架，查看其某个维度的属性特征，如图 2.21 所示。

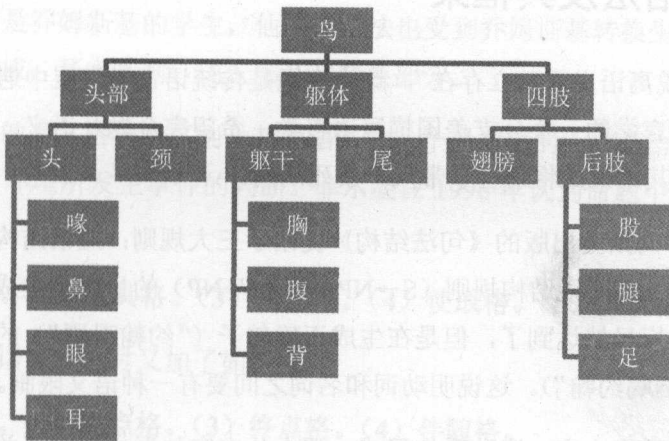


图 2.21 “鸟”的属性特征

激活“鸟”的本体框架，查看其行为特征，如图 2.22 所示。

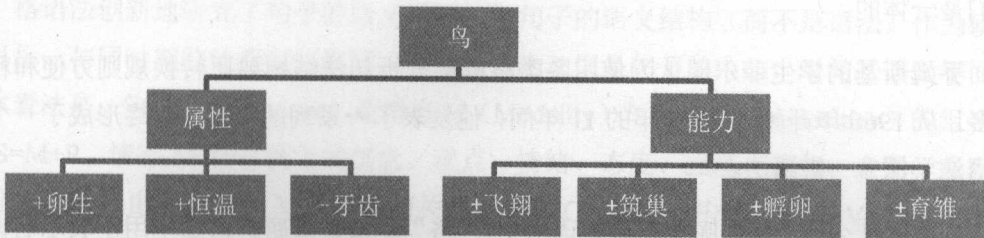


图 2.22 “鸟”的行为特征

如图 2.19~图 2.22 所示,这样就构建了一个关于鸟的最基本的本体知识库。对于“动物园里有许许多多的鸟”这句话,我们知道,在“动物园”这个本体中,存在着很多的“鸟”本体的个体。

通过激活“鸟”这个类(本体),可以查看到“鸟”类的上位节点都有“动物”→“生物”→“具体物”……;还可以看到“鸟”的兄弟节点有“兽”、“鱼”、“虫”;其下位节点有“鹰”、“雕”、“黄莺”、“天鹅”、“孔雀”、“鹦鹉”、“鸳鸯”等。

通过查看“鸟”类的属性节点,可以看到“头部”、“躯体”、“四肢”等。通过“鸟”类的行为特征,可以看到鸟都是“卵生”的“恒温”动物(+),但都没有牙齿(-)。有的鸟会“飞翔”,但有的却不会(±)。“筑巢”、“孵卵”和“育雏”的情况也都如此。

这样就建立起了一个最基本的概念网络,或者说一个关于“鸟”的完整的本体结构。关于知识库的研究,本节只是起了个头,在第 7 章以 HowNet 为例,还会详细讲解。

2.5.2 格语法及其框架

“语法能否脱离语义而独立存在”,这个问题是传统语言学发展中遇到的一个重要问题,结构主义语言学的一个分支美国描写语言学,希望完全抛开语义,独立研究语言的形式——语法。其结果是将语言学带入一条死胡同。

1957 年,乔姆斯基出版的《句法结构》提出了三大规则:短语结构规则、转换规则和语素音位规则。其短语结构规则($S \rightarrow NP+VP$; $V+NP$)的目标是生成所有的句子。生成所有句子的目标虽然达到了,但是在生成正确句子(“约翰喝酒”)的同时,也生成了错误的句子(“酒喝约翰”)。这说明动词和名词之间要有一种语义限制。

如果把语义看作内容,语法看作形式,那么从内容与形式的辩证关系来看,没有内容,形式就无法存在;没有形式,内容也无从表现。这两者是相互依赖、相互制约,各以对方为存在条件的。我们只能在抽象的意义上把它们区别开,而在具体的语言实例中它们是一体的。

乔姆斯基的学生菲尔墨认为使用各类格框架分析句法结构要比转换规则方便和精密得多。从 1966 年开始到 1977 年的 11 年间,他发表了一系列的论文,最终形成了一个新的语法学派——格语法。

那么,什么是“格”呢?在传统语法中,“格”是指某些屈折语法中用于表示名词和代词的形态变化,这种语法格都有显性的形态标记。印欧语共有 8 个格:主格、属格、

与格、宾格、离格、工具格、位格和呼格。其中,英语有3个格:主格、宾格和所有格(属格)。在汉语中,名词和代词没有形态变化,所以没有语法格。传统语言学中所说的格只是表层格。

然而,“格”语法所说的格不是这个含义。它是指句子中的体词(名词、代词等)和谓词(动词、形容词等)之间的及物性关系(Transitivity)。这些关系表示为一种语义关系,也称为“深层格”。称其为深层格的原因是,一旦句子中的谓词与其搭配的名词或代词被确定下来,也就是说谓词的语义格被确定下来(注意,一个谓词可以有多种语义格,这里的“确定”是指选定了其中的一种),那么,句子的语义结构也就确定了,而不管句式发生哪种变化,即它们与谓词形成什么语法关系,或者表层结构经过何种转换操作,即变换何种位置,整个句子的语义都不发生变化。也就是说深层格(语义格)与语言表层结构的语法概念都没有对应关系。更重要的是,格语法可以跨越不同的语言,是一切语言中普遍存在的现象。对于汉语这种语义为先的语言而言这个规律尤其重要,可以理解为三个平面中语义的先决性。

由于菲尔墨是乔姆斯基的学生,他的格语法也受到乔姆斯基转换生成语法的影响,因而由三部分组成:基本规则、词汇部分和转换部分。

本文要强调的是,格语法区别于其他语法的一个最重要的贡献是给出了格表。深层格是人类对周围环境所发生事件的判断。菲尔墨在1996年认为命题中需用的格包括如下6种。

(1) 施事格。(2) 工具格。(3) 承受格。(4) 使成格。(5) 方位格。(6) 客体格。

后来,他在语言分析时又加了如下格。

(1) 受益格。(2) 源点格。(3) 终点格。(4) 伴随格。

格表对之后的语义角色标注理论产生了直接的影响,最终成为句子语义解析的通用数据类型。

格语法创新地研究了句子的语义平面,把句子的语义结构(而不是语法)作为研究的目标,与同时期的依存句法相同,菲尔墨认为动词是句子的中心,对句子语义平面的基本看法是:句子S(Sentence)是情态M(Modality)与命题P(Proposition)的组合,即 $S=M+P$ 。情态M指言谈者的信念、观点、情绪、态度、立场、语气、意图、观察角度等。命题P由一个动词V及与其相关的多个格C(Case)构成,即 $P=V+C_1+\cdots+C_n$ 。同时,菲尔墨认为词库是语言中词汇的集合,在词库中除要标明每一个词条在句法、语义和语音方面的特征外,还需标明它们的底层格的特征。

菲尔墨主要研究了有关格的形式和主语确定的转换规则。他认为深层格所体现的语义关系是一个固定而统一的概念，而在表层结构中的表现形式则因语言而异。有些语言主要通过介词来表现，有些语言用屈折变化和词汇变化来表现，有些语言则主要采用次序来表现，有些语言综合采用上述各种形式。

“格”语法理论对描写语法学来说是有用的，但是也有其局限性。我们知道，句子中的语义关系是复杂、多样的，而“格”语法理论只适用于说明名词和动词之间的语义关系，对于其他语义关系还未涉及。虽然理论存在局限，但它给人以启迪。正是在“格”理论的影响下，汉语语法研究中出现了崭新的分析手段——语义角色的研究，以适用于复杂、多样的语义关系。

2.6 结语

在汉语自然语言处理过程中，遇到的问题与印欧语言有诸多不同：首先中文分词就是西方语言所不曾遇到的问题；在句法解析环节，如果照搬西方的算法理论，则越来越多的事实证明，句法分析在汉语上的精度都要显著低于西方语言。如何解决这些差异性要求我们必须深入研究汉语本身，我们希望对汉语历史的研究，来更深刻地认识汉语的普遍性和特殊性。

本书是研究汉语语言处理的一本专业书籍。本章回顾了汉语语言学研究的一些成果，目的就是使读者更加深入地了解汉语的起源、发展及成熟几个阶段的流变。

本章的主体内容共分为三部分：文字的起源、发展与流变；句子、词汇构成与白话文；语义研究的基础。

文字的起源、发展与流变揭示了文字符号的来源、特别对古代造字的 6 种方法——六书，从认知科学的角度给出了解释，使读者从一个新的角度来看待汉字的起源。在字形的流变一节，旁征博引地给出了汉字符号化的整个过程，包括历史的外因、隶变的方式，以及符号化后汉字各种构成的细节。

句子、词汇构成与白话文详细分析了只有句子才是构成完整语义的基本形式。然后给出了汉语早期的语言形式，并揭示了最初语言形式、思维形式和逻辑形式的三位一体性的基本规律。

接下来,深入探讨了复音词的来历:从古至今两千年间,复音词的变化、发展,以及早期白话文的产生。最后,简要回顾了白话文的句法研究。

本章的最后介绍了汉语三个平面中语义研究的一些基础理论:词汇的语义研究最终发展出了本体论;格语法及其框架最终发展出了谓词论元的概念。由于这些内容在后面的章节中也有涉及,这里仅仅给出了一些介绍的文字,以供读者参考。

本姓同食己其同

第 3 章

词汇与分词技术

本章讲解汉语自然语言处理的第一项核心技术：中文（或汉语）词汇自动切分，也称为中文分词技术。从 1979 年，中国就开始进行机器可读语料库的建设，专业的高校和研究机构纷纷建立大规模中文语料库。这个阶段历经十多年之久，由于语料库建设之初，许多工作都要从零开始，分词任务都由专业人员手工完成。这是一项繁重而枯燥的工作。即便如此，受到人为主观因素的影响，人工分词的标准并不统一，语料精度也不高。虽然是国家级的项目，所谓的“大规模”语料库的规模不过也就是百万级。因此，迫切需要统一的分词规范及适合大规模语料的高精度中文分词算法。

中文分词的研究经历了二十来年，现在看来基本上分为如下三个流派。

- 机械式分词法（基于词典）。机械分词的原理是将文档中的字符串与词典中的词条进行逐一匹配，如果词典中找到某个字符串，则匹配成功，可以切分，否则不予切分。基于词典的机械分词法，实现简单、实用性强，但机械分词法的最大的缺点就是词典的完备性不能得到保证。据统计，用一个含有 70 000 个词的词典去切分含有 15 000 个词的语料库，仍然有 30% 以上的词条没有被分出来，也就是说有 4 500 个词没有在词典中登录。
- 基于语法和规则的分词法。其基本思想就是在分词的同时进行句法、语义分析，利用句法信息和语义信息来进行词性标注，以解决分词歧义现象。因为现有的语法知识、句法规则十分笼统、复杂，基于语法和规则的分词法所能达到的精确度还远远不能令人满意。目前这种分词系统还处在试验阶段。

- 基于统计的分词法。其基本原理是根据字符串在语料库中出现的统计频率来决定其是否构成词。词是字的组合，相邻的字同时出现的次数越多，就越有可能构成一个词。因此，字与字相邻共现的频率或概率能够较好地反映它们成为词的可信度。

针对这些问题，经过 20 年来的不懈努力，最终较成功地实现了中文词汇的自动切分技术。本章简要介绍了 ICTCLAS 中文分词算法的来源和现状，以及实现了 NShort 最短路径算法的一些著名的开源框架。为了读者理解方便，我们选择 HaNLP 系统提供的开源框架，结合实例，详细分析和讲解著名的 NShort 最短路径分词方法。内容包括：一元词网与原子切分、生成二元词图、NShort 最短路径、命名实体识别、细分阶段等内容。

迄今为止，该算法仍旧是中文自然语言处理的最大突破和最重要的成果。

3.1 中文分词

信息处理的目标是使用机器（主要指计算机）能够理解和产生自然语言。而自然语言理解和产生的前提是对语言能够做出全面的解析。汉语词汇是语言中能够独立运用的最小的语言单位，是语言中的原子结构。因此，对中文进行分词就显得至关重要。

前面讲过，汉字起源于象形字发展起来的表意符号（部分也表音，但不精确）。从古至今每个汉字之间都是相互独立的结构，在早期的文言文中，一个汉字表达一个完整的语义，一个汉字就是一个词，所以不存在分词的问题。随着白话文的发展，词汇从单音词走向了复音词，从独字为词发展为以二字词为主，独字词、多字词并存的阶段。以复音词占主体的现代白话文，却没有明确地给出划分词汇的标志。这与千百年来中文行文习惯上不实行分词连写有关。无论是文言文还是白话文，为了明确语义，仅在句子停顿处标以句读，这是古已有之的，但当时的标点符号不规范、不统一，你标你的，我标我的，各自为政，没有系统地发展起来。现代标点符号系统源于五四运动，之后几十年不断发展为白话文的统一书写规范，虽然它反映了现代汉语的进步，但这些符号主要都是标句符号，对于复音词之间的划分并未区分。

另一个重要方面，因为文化的巨变，从五四运动确立了白话文作为通用的书面语，距今时间还不到一百年。这近百年来，在语法、语义方面的研究也走了不少弯路。很多理论还不够成熟，独立的计算语言体系也未建立。基本上，中文的实际应用在传统与西方的双重影响下此消彼长地发展，还处于一个未成熟的阶段。

3.1.1 什么是词与分词规范

1996 年,有人通过 6 个母语为汉语的被试者对同一篇文本进行手工分词,文本由 100 个句子组成,含 4 372 个字。被试者共 6 人,其中三位来自内地,三位来自我国台湾地区。人们因自身居住地区的不同、受教育程度等因素,对词汇的切分产生的差异,称为词语认同率。与人们的猜测有很大的不同,两个地区的人对词汇的认同产生了比较大的差异。其中,差异最大的仅为 0.69,最小为 0.89,平均认同率为 0.76。((《中文分词十年回顾》))由此可见,即便是母语为汉语的使用者,对于划分词汇的标准也是有分歧的。

那么,什么是词,我们如何界定汉语词呢?古往今来,汉字虽然有 5 万多个,但常用的汉字大约仅有 6 000 个。即便如此,其中很多汉字在日常生活中较少用到。然而,这些有限的汉字足以维持词汇的长期更新,因为扩大中文词汇的方法是通过构造汉字的复合新词,而不是创造新的字符来完成的。这就造成了汉语中所谓的词和短语之间没有明确的界限。这可能也就是中国的一些语法学家认为,中文没有词语而只有汉字的原因,并创造了一个术语——“字短语”来代替传统的词汇。董振东就认为:“‘词或字符’的争论源于他们都急于给中国语言一个硬规范的共同基础。遗憾的是,中文不是那么明确的或硬的,它是软的。我们必须认识到其‘柔软度’”。

除构词法的原因之外,人们还因为自身的方言、受教育程度、亚文化等差异因素,对词汇的认识也不同。这些事实都成为制定统一的汉语分词标准的障碍。但是对于计算机系统而言,同一种语言应使用统一的规范来划分词汇,否则,很难想象在多重标准下的词汇系统能够计算出相同的语义结果。这是计算机系统所提供的规定性。因此,构建一套统一的分词规范就显得尤为重要。

随着 NLP 的大规模应用,计算语言学界逐渐统一了汉语词汇的标准。从最初的“结合紧密,使用稳定”到信息处理领域的《信息处理用现代汉语分词规范》的制定,都是确定汉语分词标准的一种尝试,该文关于汉语词的定义给出了如下说明。

汉语信息处理使用的、具有确定的语义或语法功能的基本单位。……

——《信息处理用现代汉语分词规范》

从计算语言学的角度来看,如果把一个句子理解为一个特殊的可计算的逻辑表达式,那么句子中的一个词就是表达式中的一个可计算符号,有的表示为连接的符号,如连词“然后”、“而且”这样的虚词;有的表示为动作、状态(函数的签名),如“出现”、“思考”等这样的动词;有的表示为事物的概念,如“中国”、“泰山”等这样的名词。

本书涉及的分词规范有如下两大类：第一类包括《北大（中科院）词性标注》、《现代汉语语料库加工规范——词语切分与词性标注》、《北京大学现代汉语语料库基本加工规范》三篇文章，读者可从 <http://www.threedweb.cn/thread-1584-1-1.html>、<http://www.threedweb.cn/thread-437-1-2.html> 下载；第二类为《宾州树库中文分词规范》，读者可从 <http://www.threedweb.cn/thread-1478-1-1.html> 下载。

本节主要介绍中文分词中最常用的《北大（中科院）词性标注》（以下简称《北大规范》）的基本原则。

《信息处理用现代汉语分词规范》和传统的语法教育中将汉语的词类主要分为 13 种：名词、动词、代词、形容词、数词、量词、副词、介词、连词、助词、语气词、叹词和象声词。这与朱德熙先生提出的 19 种分类法有所不同。朱先生的分类法还包括：时间词、处所词、方位词、区别词、状态词等。除此之外，《北大规范》还加入了 4 个兼类谓词：副动词、名动词、副形词、名形词；最后还增加了前缀、后缀、成语、简称、习用语 5 种辅助词类。这样，《北大规范》就形成了 40 种词类。

这 40 种词类与《信息处理用现代汉语分词规范》所述的 13 种词类可以用表给出对照关系（见表 3.1）。

表 3.1 修改后的现代汉语词语分类体系对照表

基本词类	实词	体词	1. 名词n	32. 人名 nr
				33. 地名 ns
				34. 机构团体 nt
				35. 其他专名 nz
		谓词	2. 时间词t	
			3. 处所词s	
			4. 方位词f	
			5. 数词m	
			6. 量词q	
			7. 代词r（体词性）	
			代词r（谓词性）	
			8. 动词v	
			9. 形容词a	
			10. 状态词z	

续表

基本词类	实词	谓词	兼类谓词	37. 副动词vd				
				38. 名动词vn				
				39. 副形词ad				
				40. 名形词an				
	虚词	11. 区别词b						
		12. 副词d						
		13. 介词p						
		14. 连词c						
		15. 助词u						
		16. 语气词y						
	17. 拟声词o							
	18. 叹词e							
附加类别	小于词的单位	19. 前接成分h		27. 名语素Ng				
		20. 后接成分k			28. 时语素tg			
		21. 语素g	29. 形语素Ag					
						30. 副语素dg		
							31. 动语素vg	
								22. 非语素字x
					其他标准			23. 成语i
	24. 习用语l							
	25. 简称略语j							
	26. 标点符号w							
36. 未知词un								

如表 3.1 所示，从分词的角度来看，这两种切分标准之间的差距在于，北大标准除考虑到词汇的语法特征之外，还兼顾了词的语义特征。但是从语义研究的角度来看，这些语义特征并不完备，不过一些更细节的词性可划归到其父类之中。

除此之外，表 3.1 中还有一个问题需要澄清，即“附加类别”中的各个子类。下面由简而繁逐个说明。

(1) 成语和习用语。这是中文的特有词汇类别，但该种方式与句子解析和词汇切分无关，本书中基本遵循这样的原则：成语基本都作为一个完整的词进行切分；习用语有的仅包含一个词，就作为词来切分，有的是一个句子，就逐词进行切分。

(2) 语素和前后缀。

- 语素。语素是构成词的最小单位，其粒度小于词汇。是否需要切分可根据用户需求，从语言处理的角度来看，语素级别的切分使用范围并不大。例如，“毛泽东”在《北大标准》中常切分为：“毛/泽东”。本书建议作为一个完整的词来对待。
- 前后缀。比较典型的前/后缀包括：初（初一）、阿（阿姨、阿爸）、老（老先生）、第（第一、第二）、儿（花儿）、们（男人们、同志们）。这要根据实际情况来做处理。其切分标准按照前/后缀的能产型和语义完整性两个标准来切分。例如，“初一”的“初”作为前缀，能产性较弱，不予切分；“们”的能产性比较强，应做切分，但也不绝对。“人们”的语义完整性更强，可不做切分。

有关更多的切分细节可参照前文给出的相应文档，并结合自身的需求，制定切分规则。

3.1.2 两种分词标准

由于语素对词汇的构成也产生影响，实际应用中，汉语分词也分为两个粒度。粗粒度分词：将词作为语言处理最小的基本单位进行切分。细粒度分词：不仅对词汇进行切分，也要对词汇内部的语素进行切分。

例如，原始串：浙江大学坐落在西湖旁边。

粗粒度：浙江大学/坐落/在/西湖/旁边。

细粒度：浙江/大学/坐落/在/西湖/旁边。

粗粒度将“浙江大学”看作一个完整的概念，对应一个完整的词汇，进行切分。而细粒度则不同，除将“浙江大学”完整切分出来之外，还要将构成“浙江大学”的各个语素切分出来：浙江/大学。

常见的例子还有很多，如“中华人民共和国”，粗粒度的分词就是“中华人民共和国”，细粒度的分词可能是“中华/人民/共和国”。一般细粒度切分的对象都为专有名词。因为专有名词常表现为几个一般名词的合成。

在实践中，粗粒度切分和细粒度切分都有其使用的范围。粗粒度切分主要用于自然语言处理的各种应用；而细粒度分词最常用的领域是搜索引擎。一种常用的方案是，在

索引的时候使用细粒度的分词以保证召回,在查询的时候使用粗粒度的分词以保证精度。在本书中,如果未加特别的说明,则都为粗粒度分词。

3.1.3 歧义、机械分词、语言模型

现代汉语的复音词结构,使少量的字符(4 000 多个)通过排列组合来表示大量的词汇(100 万个以上),而中间又没有标点的分隔,最容易出现的问题是歧义问题。歧义问题在汉语中普遍存在,长久以来歧义切分问题一直是中文分词的核心问题之一。对此,梁南元等已经做过广泛和深入的研究。下面给出几种重要的歧义切分的研究成果。

定义 7-1 (交集型切分歧义) 汉字串 AJB 称作交集型切分歧义,如果满足 AJ 、 JB 同时为词(A 、 J 、 B 分别为汉字串),则此时汉字串 J 称作交集串。(梁南元 1987)

例如,交集型切分歧义:“结合成”。

其中, A = “结”, J = “合”, B = “成”。

一种切分为:(a) 结合 | 成; 另一种切分为:(b) 结 | 合成

这种情况在汉语文本中非常普遍,如“大学生”、“研究生物”、“从小学起”、“为人民工作”、“中国产品质量”、“部分居民生活水平”等。为了刻画交集型歧义字段的复杂结构,梁南元还定义了链长的概念。

定义 7-3 (组合型切分歧义) 汉字串 AB 称作多义组合型切分歧义,如果满足 A 、 B 、 AB 同时为词。

例如,多义组合型切分歧义:“起身”。在如下两个例子中,“起身”有两种不同的切分:(a) 他站 | 起 | 身 | 来。(b) 他明天 | 起身 | 去北京。类似的,“将来”、“现在”、“才能”、“学生会”等,都是组合型切分歧义字段。

梁南元(1987a)曾经对一个含有 48 092 字的自然科学、社会科学样本进行了统计,结果交集型切分歧义有 518 个,多义组合型切分歧义有 42 个。据此推断,中文文本中切分歧义的出现频度约为 1.2 次/100 字,交集型切分歧义与多义组合型切分歧义的出现比例约为 12:1。

有意思的是,据文献[刘挺等,1998a]的调查却显示了与梁南元截然相反的结果:汉语文本中交集型切分歧义与多义组合型切分歧义的出现比例约为 1:22。孙茂松认为,造成这种情形的原因在于,定义 7-3 有疏漏。因此,孙茂松等(2001)曾经猜测,加上一条上下文语境限制才真正反映了梁南元的本意。

定义 7-3' (多义组合型切分歧义) 汉字串 AB 称作多义组合型切分歧义, 如果满足 (1) A、B、AB 同时为词; (2) 文本中至少存在一个上下文语境 c, 在 c 的约束下, A、B 在语法和语义上都成立。

——上文均来自《统计自然语言处理》—宗成庆著

针对上述问题, 人们设计了早期的机械分词系统。机械分词系统都是基于最大匹配方法作为最基本的分词算法。该方法由苏联汉俄翻译学者提出, 也称为 MM (The Maximum Matching Method) 方法。其基本思想是, 假设自动分词词典中的最长词条所含汉字个数为 I, 则取被处理材料当前字符串序数中的 1 个字作为匹配字段, 查找分词词典。若词典中有这样的一个 I 字词, 则匹配成功, 匹配字段作为一个完整的词被切分出来; 如果词典中找不到这样的一个 I 字词, 则匹配失败。匹配字段去掉最后一个汉字, 剩下的字符作为新的匹配字段, 回到上述步骤, 重新匹配, 如此进行下去; 直至切分到成功为止。即完成一轮匹配, 并切分出一个词, 之后再按上述步骤进行下去, 直到切分出所有词为止。

例如, 现有短语“计算机科学和工程”, 假设词典中最长词为 7 字词, 于是先取“计算机科学和工”为匹配字段, 来匹配词典, 由于词典中没有该词, 故匹配失败; 去掉最后一个汉字成为“计算机科学和”作为新的匹配字段, 重新匹配词典, 同样匹配失败; 取“计算机科学”作为新的字段来匹配词典, 由于词典中有“计算机科学”一词, 从而匹配成功, 切分出的第一个词为“计算机科学”。以此类推, 直至切分出第二、三……个词。

使用 MM 方法切分的精度并不高, 很难达到实际应用的要求, 随着语料的增大, 误差也逐渐变大。之后人们又基于此方法提出了双向匹配法。该方法是从最大匹配方法发展而来的, 分为正向最佳匹配法和逆向最佳匹配法。它们的基本原理都是相似的: 将待分析的汉字串与机器词典中的词条进行最大匹配, 若在词典中找到某个字符串, 则匹配成功 (识别出一个词)。所不同的是, 两个算法的搜索方向相反。待处理的字符串中存在着交叉歧义, 因此两种方法所得的结果必然不同。当然, 基于最大匹配的搜索方法还存在着局限性, 比如正向最大匹配, 因为只能正向地找出最长的词, 而不能找出所有的候选词条。因此, 后来发展出了双向扫描法来更快速地检测出歧义产生的位置。

这类早期的分词器因为没有考虑到词汇上下文的相关性, 分词的准确度都不高。基于正向最大匹配算法的分词器的准确度为 78%; 召回率为 75%; F1 值约为 76%。后来改进的双向匹配算法的最高精度也在 80% 左右徘徊。显然这不能满足高精度文本处理的需求。

基于机械方法的分词器虽然没有得到广泛的应用,但是却揭示了一个重要的语言规律:一个词汇的出现与其上下文环境中出现的词汇序列存在着紧密的关系,如果算法不能反映和处理这种上下文依赖关系,则不能最终达到满意的分词结果。所谓上下文相关性是指,文本中第 n 个词的出现与其前后 $n-m$ 到 $n+m$ 个词有高度的相关性,而与这个范围之外的其他词的相关性较低。我们把 $[-m, m]$ 范围也称为窗口范围。

考虑单侧的情况,文本中第 i 个词的出现与其前面 $i-n$ 个词相关 ($0 < n < i$),而不考虑这个窗口之外的其他词的相关性。形式化如下,假设 $W_1 W_2 \cdots W_n$ 是长度为 n 的字串,则词 W_i 出现的似然度用方程表示为:

$$\begin{aligned} P(W) &= P(W_1 W_2 \cdots W_n) \\ &= P(W_1) P(W_2 | W_1) P(W_3 | W_1 W_2) \cdots P(W_n | W_1 W_2 \cdots W_{n-1}) \\ &= \prod_{i=1}^n P(W_i | W_1 W_2 \cdots W_{i-1}) \end{aligned}$$

由上式可知,要计算得到词 W_i 的出现概率,必须要知道在 W_i 之前所有词的出现概率。这显然不可行,也没有意义。但是如果任意一个词 W_i 的出现概率只与它相邻的一个词有关,那么问题的解决即可得到极大的简化。这时的语言模型叫作二元模型,也称为一阶 Markov 链,即:

$$\begin{aligned} P(W) &\approx P(W_1) P(W_2 | W_1) P(W_3 | W_2) \cdots P(W_n | W_{n-1}) \\ &= \prod_{i=1}^n P(W_i | W_{i-1}) \end{aligned}$$

当 $n=2$ 时,统计二元概率的计算公式就为:

$$\prod_{i=1}^n P(W_i | W_{i-1}) \approx \frac{\text{Count}(W_{i-1} | W_i)}{\text{Count}(W_{i-1})}$$

$\text{Count}(\cdot)$ 表示一个特定词序列在整个语料库中出现的累计次数。

将语言模型应用到分词算法中,中文分词的水平将会得到显著的改善,ICTCLAS 的中文分词算法就是这方面的最成功的案例,经测试该分词器的准确率为 98%;召回率为 98.50%; $F1$ 值约为 98%。这一改善与之前的分词系统相比有一个显著的质变。该算法使高精度中文文本处理成为可能。

3.1.4 词汇的构成与未登录词

能够正确地消除切分中的歧义,使计算机处理词汇问题又前进了一大步。但是问题仍旧没有完全解决。所谓词汇,一般都具有三个重要的特性:稳固性、常用性和能产性。稳固性、常用性都比较容易理解。关键问题在于能产性。前面讲过有关汉语复音词的构词方法,与拼音文字不同,汉语的构词机制是一个动态的自组织认知系统,而不是一个静态的系统。随着外界新事物的产生,表达概念的新词汇也会层出不穷地涌现,这不是什么特别的技能,对于中国人而言是一种本能,几乎人人都具备。人们给小孩子起名字就是词汇的能产性生动的体现:一方面,孩子的名字要继承父辈的姓氏,另一方面又要传达大人对孩子未来最美好的期望,而又尽可能不与他人的名字重复。这个过程就是词汇能产性的表现。网络媒体、专业术语、组织机构的命名都是新词产生的重要来源。

在自然语言处理中,它们被统称为未登录词识别(Named Entity Recognition, NER)。在真实文本的切分中,未登录词总数的大约九成是专有名词,其余的为通用新词或专业术语。因此,未登录词识别就是包括中国人名、译名、日本人名、地理位置名称、组织机构名称等专有名词的识别。在自然语言处理研究中,人们通常将上述专有名词和数字、日期等词称为命名实体。由于命名实体识别不仅是汉语自动分词研究中的关键问题,也是诸如英语等其他语言处理中的重要问题,它的处理效果直接影响到信息抽取、信息检索和机器翻译、文摘自动生成等应用系统的性能。因此,近几年来专有名词的处理(包括识别、翻译等)已经成为自然语言处理研究中一个非常活跃的分支。

以往人们的处理方法常从构词学的角度来研究算法,因此研发了很复杂的基于构词编码的方法(下文介绍的 HanLP 中文分词的系统就是一种基于构词角度来进行命名实体识别的算法)。事实证明,这种做法对于狭窄的专门领域(人名中的中国人名、译名、日本人名)等的未登录词识别,能够获得较好的效果,但在处理大规模不同领域的未登录词问题上存在着很大的障碍,至少是不现实的。

笔者认为应从语义类的角度来重新考虑这个问题。不同语义类下的未登录词,在统计学规律上具有相似性。利用这一点,近些年命名实体识别方面新的算法层出不穷,已经证明,基于半监督的条件随机场(semi-CRF)算法,对于处理不同领域的专名识别具有较低的成本和较好的效果。

3.2 系统总体流程与词典结构

3.2.1 概述

ICTCLAS 汉语分词系统是由张华平博士于 2002 年设计开发的一套中文分词系统。迄今为止, ICTCLAS 都是自然语言处理领域国人设计的人工智能类算法中最成功、应用最广泛的成果。该算法具有原理简单, 易于实现, 便于使用, 以及对于大规模分词, 效率高、精度准等优势。除此之外, 该算法使用的模型具有资源占用低、便于增量扩展等特点。

在自然语言处理领域, 一个成功的算法就意味着一个新时代的到来。首先, 该算法一经问世就取得了业界的广泛关注, 并斩获诸多奖项, 获 2002 年国内 973 评测综合第一名; 在 2003 年国际 SIGHAN 分词大赛中获得综合评比第一名; 2010 年获得了钱伟长中文信息处理科学技术奖一等奖。目前, 据笔者所知, 各大知名的网络公司都以该算法作为分词系统的核心算法。该算法使中文自然语言处理终于从词法分析时代跨入了句法分析时代。

ICTCLAS 的商业版经过张博士倾力十余年的精心打造, 内核升级 10 余次, 已经成为精度最高、速度最快的中文分词系统。该系统还提供一个基于 30 万个常用词的免费版, 可从 <http://ictclas.nlpir.org/> 下载。该网站也称为自然语言处理与信息检索共享平台, 分词器名称调整为 NLPIR 汉语分词系统, 并逐年升级。现在的 NLPIR 汉语分词系统 (又称为 ICTCLAS 2013), 主要功能包括中文分词、词性标注、命名实体识别、用户词典功能, 以及支持 GBK 编码、UTF8 编码、BIG5 编码, 还新增微博分词、新词发现与关键词提取等功能, 是国内最著名的分词系统之一。

为了促进大家的研究学习, ICTCLAS 发布一个免费开源版称为 FreeICTCLAS, 均使用 C++ 开发语言。目前, 能够找到的 FreeICTCLAS 原始版本应该有两套: 一套是基于 Windows 操作系统的; 另一套是移植到 Linux 操作系统下的版本。读者可以从 <http://www.threedweb.cn/thread-162-1-1.html> 下载 Linux 版本的; 也可以从 CSDN 上找到基于 Windows 版本的。张华平博士在设计这个分词系统时还是研究生, 因此代码具有很强的实验性, 两套版本的核心源码都完全相同, 仅支持 GBK 编码, 不支持 UTF-8 或其他编码。目前两个版本都免费提供, 以供大家学习。

笔者本来计划以此版本作为算法讲解的源码,但综合考虑到代码的易读性、实现的成熟度、编码支持及跨平台等诸多因素,决定选取较为清晰的 ICTCLAS 的重写版本来讲解。目的是有助于读者能够更清晰、深刻地了解算法的本质。目前,很多朋友用其他语言重写了 ICTCLAS,包括使用 Java 语言的 ICTCLAS4J,孙健的 Ansj(新版本算法为 CRF)及 C#的 SharpIctclas 等。其中,比较全面的是 HanLP,网址:<http://hanlp.linrunsoft.com/>。选取这个 Java 框架的原因不仅是其对 ICTCLAS 的算法还原得比较好,另外,它还是一个非常丰富的 NLP 算法库,除 N-最短路径算法之外,还实现了 HMM 的 Viterbi 算法、最大熵算法直至 CRF 算法等。同时,该算法库还提供了丰富的语料资源,供用户使用。有关更多细节读者可以下载源码学习。

3.2.2 中文分词流程

从本节开始,将详细介绍 ICTCLAS 分词算法原理,以及简单介绍系统的基本构成。这里使用的是 HanLP 分词模块,操作系统是 Linux/Windows,开发工具是 Eclipse mar 64 位 IDE。源代码可以从 <http://hanlp.linrunsoft.com/release/hanlp-1.2.8-sources.jar> 下载,词典文件的下载地址:<http://pan.baidu.com/s/1ntRkylN>。

ICTCLAS 的算法思想来源于 HMM。Java 版本的重构在功能和逻辑上与 C++版本基本相同。HanLP 的实现做了如下修改。

- ❑ 使用双数组 Tire 树读取和检索词典,提高性能。
- ❑ 使用提供了更丰富的命名实体识别库:人名、译名、日本人名、地名、组织机构名等。
- ❑ 搜索引擎系统的支持等——超出本节内容,讲解中未涉及。
- ❑ 细分阶段使用了最短路径法。

当然,系统也加入了词性标注的功能。词性标注不是本章的重点,我们留到以后讲解。为了完整地了解分词的过程我们将其分为如下两部分。第一部分是整体流程及一些重要的数据结构的讲解;第二部分是关于流程每个节点的具体实现和代码注释,以及各阶段的执行结果。

图 3.1 所示为 HanLP 整体的运行流程,也是全套文档的线索结构。在逐步了解文档内容之前,先对此流程图进行简要的介绍。

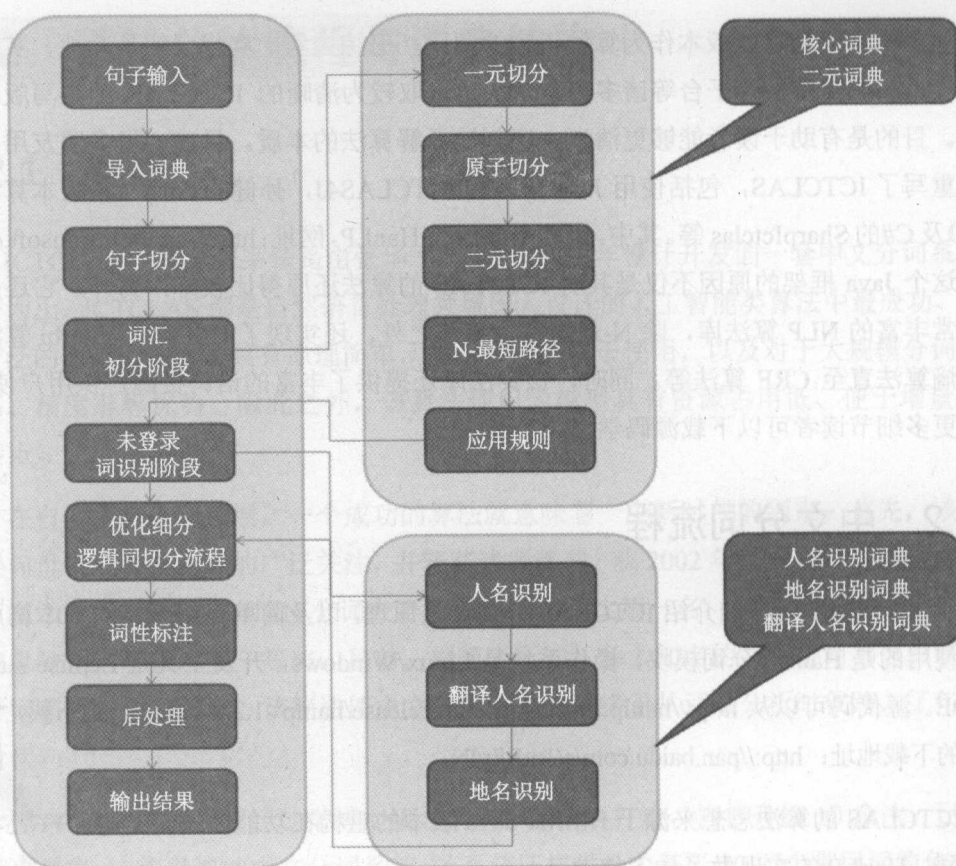


图 3.1 HanLP 整体的运行流程

(1) 在第一个环节，系统读取待分词的字符串。输入的可以是一个句子也可以是一个篇章，如果是篇章则系统会首先进行句子切分，然后调用多线程，对每个切分的子句进行分词。

(2) 根据输入的配置信息，导入相应的词典。

常用的词典可到专门的网址下载。下载地址：<http://pan.baidu.com/s/1ntRkylN>。词典一共 624.5MB，全部词典种类比较多，还包括若干语言模型。本算法所需的词典位于“data/dictionary/”目录下。

包括 CoreNatureDictionary（一元语言模型词典）、CoreNatureDictionary.ngram（二元语言模型词典）、person/nr.tr, nrf, nrj（人名识别词典—中国人名、译名、日本人名）、place/ns.tr（地名识别词典）、organization/nt.tr（组织机构名词典），它们的文件都有文本格式的副本，具体结构在 3.2.3 节中进行分析。

(3) 进入粗分阶段。

- ❑ 首先对句子进行字符级切分，即将输入的句子切分为单个 UTF-8 编码的字符数组（函数 `toCharArray()`），包括单个中文字符、单个英文字符、其他单个字符等。
- ❑ 一元切分。查询核心词典，将字符切分的结果与词典词最大匹配，匹配的结果，包括词形、词性、词频等信息形成一元词网，之后对一元词网进行原子切分，即按模式合并英文和数字的字符构成原子词。
- ❑ 二元切分。用一元分词的结果（二维数组）查询二元词典，与二元词典进行最大匹配，匹配的结果为一个 `Graph`，形成一个词图。
- ❑ NShort 算法计算。将查出的每个结果按平滑算法计算二元分词的词频数得到词图中每个节点的权值（概率的倒数），应用 NShort 算法累加词图中每个节点构成的所有路径，权值最小（概率最大）的那条路径对应的词图节点就是初分的结果。
- ❑ 对粗分结果执行后处理应用规则，识别时间类专有名词。

(4) 进入未登录词识别阶段，使用隐马尔科夫链语言模型。

- ❑ 根据人名识别词典，将粗分的结果与之匹配，Viterbi 算法识别外国的人名。
- ❑ 根据地名识别词典，将粗分的结果与之匹配，Viterbi 算法识别地名。
- ❑ 根据组织机构名词典，将粗分的结果与之匹配，Dijkstra 算法识别组织机构名。

(5) 将命名实体识别后的分词结果加入词图中，对词图再次进行分词（Dijkstra 最短路径法）。该阶段为细分阶段。

(6) 使用词性标注模型，Viterbi 算法，对分词结果进行词性标注；该部分将在第 4 章介绍。

(7) 转换路径为分词结果，并输出分词的结果。

如图 3.1 所示，整个流程的主程序位于 `NShortSegment` 类的 `segSentence` 方法内。它处于句子切分阶段之后。此函数给出了整个分词流程的总过程，其代码如下。

```
@Override
public List<Term> segSentence(char[] sentence) {
    WordNet wordNetOptimum = new WordNet(sentence); // 最优词网
    WordNet wordNetAll = new WordNet(sentence);
    // 1. 粗分阶段
    List<List<Vertex>> coarseResult = BiSegment(sentence, 2, wordNetOptimum,
wordNetAll);
```

```

boolean NERexists = false;
for (List<Vertex> vertexList : coarseResult) {
    if (HanLP.Config.DEBUG) {
        System.out.println("粗分结果" + convert(vertexList, false));
    }
    // 2. 实体命名识别阶段
    if (config.ner) {
        wordNetOptimum.addAll(vertexList);
        int preSize = wordNetOptimum.size();
        if (config.nameRecognize) {
            PersonRecognition.Recognition(vertexList, wordNetOptimum, wordNetAll);
        }
        if (config.translatedNameRecognize) {
            TranslatedPersonRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
        }
        if (config.japaneseNameRecognize) {
            JapanesePersonRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
        }
        if (config.placeRecognize) {
            PlaceRecognition.Recognition(vertexList, wordNetOptimum, wordNetAll);
        }
        if (config.organizationRecognize) {
            // 层叠隐马模型——生成输出作为下一级隐马输入
            vertexList = Dijkstra.compute(GenerateBiGraph(wordNetOptimum));
            wordNetOptimum.addAll(vertexList);
            OrganizationRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
        }
        if (!NERexists && preSize != wordNetOptimum.size()) {
            NERexists = true;
        }
    }
}
// 3. 细分阶段
List<Vertex> vertexList = coarseResult.get(0);
if (NERexists) {
    Graph graph = GenerateBiGraph(wordNetOptimum);
    vertexList = Dijkstra.compute(graph);
    if (HanLP.Config.DEBUG) {
        System.out.printf("细分词网: \n%s\n", wordNetOptimum);
    }
}

```



```

        System.out.printf("细分词图: %s\n", graph.printByTo());
    }
}
// 数字识别
if (config.numberQuantifierRecognize) {
    mergeNumberQuantifier(vertexList, wordNetAll, config);
}
// 如果是索引模式则全切分
if (config.indexMode) {
    return decorateResultForIndexMode(vertexList, wordNetAll);
}
// 4. 词性标准阶段
if (config.speechTagging) {
    speechTagging(vertexList);
}
return convert(vertexList, config.offset);
}

```

上述代码为整个中文分词、命名实体识别、词性标注的全过程。

3.2.3 分词词典结构

在进入源码分析之前，首先应该了解一下整个项目所用的词典。词典是中文分词重要的外部资源。无论是标准的中文分词还是命名实体识别都离不开词典提供的词汇和语言模型资源。

系统共有 7 个不同的词典（包括语言模型）放置于 Data 目录下。HanLP 词典结构如表 3.2 所示。

表 3.2 HanLP词典结构

词典文件	说 明
CoreNatureDictionary	一元语言模型词典
CoreNatureDictionary.ngram	二元语言模型词典
person/nr. txt.trie.dat person/nr. txt.value.dat	人名识别词典—中国人名
person/nrf. txt.trie.dat person/nrf. txt.value.dat	人名识别词典—译名
person/nrj. txt.trie.dat person/nrj. txt.value.dat	人名识别词典—日本人名

续表

词典文件	说 明
place/ns. txt.trie.dat place/ns. txt.value.dat	地名识别词典
organization /nt. txt.trie.dat organization /nt. txt.value.dat	组织机构名词典

(1) 一元语言模型词典也被称为核心词典，文件名为 CoreNatureDictionary，系统提供了一个文本版本和一个二进制版本的文件。打开文本版本的文件，截取词典的片段内容如下。

```
跳梁小丑  nz  16
跳棋  n  1.
跳楼  v  136
跳楼价  nz  6
跳楼秀  nz  1
跳槽  vi  71  vn  55
跳水  vn  137
跳水教练  nnd 1
```

一元语言模型词典的第一列是词，第二列是该词的第一词性，第三列是对应该词性的词频；如果存在第四列，则是对应该词的第二词性，第五列是对应第二词性的词频；之后以此类推。

CoreNatureDictionary 是算法的核心词典，其他各个词典都从此词典衍生而来。而且，一元语言模型词典的大小代表了系统的规模，即这个词典越大，所包含的词汇就越多，能够正确分词的语料范围就越大。一般可用的、最小规模的中文分词器，其核心词典的规模不能少于 30 万词。

(2) 二元语言模型词典的文件名为 CoreNatureDictionary.ngram，系统提供了一个文本版本和一个二进制版本的文件。打开文本版本的文件，截取词典的片段内容如下。

```
像@跳梁小丑 24
像@跳舞 4
像@踢 18
像@踢皮球 2
像@踩着 2
像@身上 2
像@身份证 6
像@身边 2
```

像@身高 2

像@躺在 10

二元语言模型词典的结构与一元语言模型词典的结构不同,第一列表示两个相邻词,用@进行分隔。例如,“像@跳梁小丑”表示前一个词是“像”,后一个词是“跳梁小丑”,它们用“@”连接起来。第二列是该相邻词(共现词)在语料库中出现的概率,如上例为24次。二元语言模型词典本质上构建出一个二维的矩阵,而且是一个方阵。矩阵的行和列就是一元词表的长度。第一列的前一个词相当于矩阵的行索引,后一个词相当于矩阵的列索引,取值为矩阵词频(“Frequency”)。因此,这个矩阵非零元素的数量充分展示了词汇的搭配信息。因为这个矩阵比较大,所以以一维列表的方式显示出来。

如果把一元语言模型词典和二元语言模型词典合在一起看,它们也可看作一个图(Graph)。其中,图的顶点为一元语言模型词典中的词;二元语言模型词典的每个相邻词为连接两个顶点的一条边,词频为边的权值。它们共同构成了 Graph 这个完整的数据结构(参见类 `com.hankcs.hanlp.seg.common.Graph`)。整个 NShort 算法就是对这个图计算最大概率的过程。

3.2.4 命名实体的词典结构

人名、译名、地名、组织机构名的识别词典均由两个文件构成,后缀名分别为 `trie.dat` 和 `value.dat`。下面仅以人名识别词典为例,简单讲解命名实体识别一元词典和二元词典的数据结构。后缀名为 `trie.dat` 的词典是一元词典,人名词典的截图如下。

```
郎 B 228 D 22 E 12 K 4 C 1 L 1
郑 B 4
郑 B 3505 C 21 E 20 D 5
郑国 X 84 Z 4
郑大 X 20
郑州 L 2 K 1 X 1
郑州市 K 1
郑重 L 6
```

此词典的数据结构与 `CoreNatureDictionary` 词典相同,第一列为词汇,第二列为第一个元模式标签,第三列为第一个元模式标签的词频;第四列为第二个元模式标签,第五列为第二个元模式标签的词频;之后依此类推。元模式位于 `package com.hankcs.hanlp.corpus.tag` 包的枚举列表 `NR` 中。表 3.3 给出了人名识别模式的元模式标签的含义。

表 3.3 元模式标签

元 模 式	含 义	示 例
B	姓氏	【张】华平先生
C	名1	张【华】平先生
D	名2	张华【平】先生
E	单名	张【浩】
F	人名前缀	【老】刘，【小】李
G	人名后缀	王【总】，吴【妈】
K	人名的上文	又【来到】于洪洋的家
L	人名的下文	新华社记者黄文【摄】
M	两个中国人名之间的成分	编剧邵钧林【和】稽道青说
U	人名的上文和姓成词	这里【有关】天培的壮烈
V	三字人名的末字和下文成词	龚学平等领导，邓颖【超生】前
X	姓与双名的首字成词	【王国】维
Y	姓与单名成词	【高峰】、【汪洋】
Z	双名本身成词	张【朝阳】
A	以下之外其他的角色	—
S	句子的开头	—

表 3.3 列出了构成人名词性匹配的元模式，所有的人名模式都是由这些元模式经过排列组合而构成的。

当使用原子字符串查询 trie.dat 时，就返回了相应的元模式及对应此元模式的概率信息，如“王”、“菲”这两个连续的原子词，查询 trie.dat 词典，可以得到相应的一个元模式：“B”、“E”。这样就构成了隐马尔科夫链的转移概率矩阵。后缀名为 value.dat 的词典是二元词典。表 3.4 所示为人名识别词典的二元词典截图，该词典在目录下有 txt 格式的文本文件。

表 3.4 人名识别词典的二元词典截图

	A	B	C	D	E	F	G	K	L	M	S	U	V	X	Z
A	20 637 728	10 000	0	0	0	0	0	195 129	0	0	0	453	0	0	0
B	0	1	155 650	0	94 221	0	2 964	0	19	0	0	24	0	0	43 757
C	0	0	0	155 393	0	0	0	0	0	0	0	47	461	0	0
D	2 409	3 427	0	0	0	0	0	0	141 568	16 990	0	0	0	63	0
E	1 491	2 198	0	0	1 000	0	0	0	81 654	9 104	0	4	0	45	0
F	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0

续表

	A	B	C	D	E	F	G	K	L	M	S	U	V	X	Z
G	6	0	0	1 000	0	0	0	0	2 907	1000	0	0	0	0	0
K	0	217 510	0	0	0	18	0	0	0	0	0	0	0	7506	11
L	256 487	1 000	0	0	0	0	0	8 039	0	0	0	7	0	0	0
M	0	29 629	0	0	0	0	0	0	0	0	0	0	0	526	0
S	1 097 701	43 031	0	0	1 000	2		21 858	0	0	0	32	0	928	1
U	0	0	251	0	275	0	0	0	0	0	0	0	0	0	46
V	447	13	0	0	0	0	0	0	0	0	0	0	0	0	0
X	0	0	0	9 064	0	0	0	0	0	0	0	5	18	0	0
Z	594	807	0	0	0	0	0	0	38 385	4010	0	0	0	19	0

表 3.4 中,第一行和第一列就是标签的序号。具体内容对照表 3.3。它们构成一个 15×15 的方阵。方阵的每个元素表示以第一个原子词的标签为序号和以第二个原子词的标签为序号组合构成中国人名的词频(概率)。对于各类人名,系统还提供了各类组合模式串。组织机构的模式串共有 36 种。这些元模式构成的人名模式保存在 `com.hanks.hanlp.dictionary.nr` 的枚举 `NRPattern` 中。由元模式构成的人名模式如表 3.5 所示。

表 3.5 由元模式构成的人名模式

名字模式编码	含 义
BBCD	姓+姓+名1+名2
BBE	姓+姓+单名
BBZ	姓+姓+双名成词
BCD	姓+名1+名2
BEE	姓+单名+单名
BE	姓+单名
BC	姓+名1
BEC	姓+单名+名1
BG	姓+后缀
DG	名2+后缀
EG	单名+后缀
BXD	姓+姓双名首字成词+双名末字
BZ	姓+双名本身成词
EE	单名+单名
FE	前缀+单名
FC	前缀+名1
FB	前缀+姓氏

续表

名字模式编码	含 义
FG	前缀+后缀
Y	姓与单名成词
XD	姓与双名的首字成词+名2

表 3.5 中，第一列是由模式元构成的模式规则；第二列是模式规则的含义。从元模式到人名模式需要使用一个 `parsePattern` 函数进行解析，该函数位于 `com.hankcs.hanlp.dictionary.nr.PersonDictionary` 类中。有兴趣的读者可以做进一步的研究。

其他列表命名实体标签同样位于 `package com.hankcs.hanlp.corpus.tag` 包的枚举列表 `NT`、`NS`、`Nature` 中。有关翻译人名识别词典和地名识别词典等数据结构基本与人名识别词典都有类似的模式标签，感兴趣的读者可以自己打开相应的文件进行研究，自行解析一下相关文件，受篇幅所限这里不再赘述。

3.2.5 词典的存储结构

HanLP 在分词过程中使用了多本词典，这些词典均在调用时才加载到内存中，其都属于即插即用型。这样做的好处在于，对于那些在算法中可选的功能和词典，没有必要都导入内存、占用空间。根据配置的不同可以选择导入不同的词典。

本节以核心词典为例，简要讲解核心词典的加载过程及存储结构。核心词典的加载是在构造 `CoreDictionary` 对象时完成的。但是，加载过程有点特殊，是通过静态代码段自动完成的。静态代码段调用了 `load` 方法负责加载词典文件到内存。代码片段如下。

```
private static boolean load(String path)
{
    logger.info("核心词典开始加载:" + path);
    if (loadDat(path)) return true; // 加载二进制缓存词典
    TreeMap<String, CoreDictionary.Attribute> map = new TreeMap<String,
Attribute>(); //红黑树
    BufferedReader br = null;
    try
    {
        br = new BufferedReader(new InputStreamReader(new FileInputStream
(path), "UTF-8"));
        String line;
        int MAX_FREQUENCY = 0;
        long start = System.currentTimeMillis();
```



```

while ((line = br.readLine()) != null)
{
    String param[] = line.split("\\s");
    int natureCount = (param.length - 1) / 2;
    CoreDictionary.Attribute attribute = new CoreDictionary.Attribute
(natureCount);
    for (int i = 0; i < natureCount; ++i)
    {
        attribute.nature[i] = Enum.valueOf(Nature.class, param[1 + 2 *
i]); // 词性
        attribute.frequency[i] = Integer.parseInt(param[2 + 2 * i]);
// 词频
        attribute.totalFrequency += attribute.frequency[i];
// 总词频
    }
    map.put(param[0], attribute);
    MAX_FREQUENCY += attribute.totalFrequency;
}
br.close();
trie.build(map);
.....

```

加载过程一共经历了如下三步。

- (1) 查找是否存在缓存词典（二进制形式），如果存在就从缓存直接加载。
- (2) 如果不存在，就加载词典的文本形式。
- (3) 最后将文本形式的词典生成新的二进制（缓存）形式。

考察一下文本形式词典的加载过程，即可了解 HanLP 的词典存储结构。首先将其保存在 `TreeMap<String, CoreDictionary.Attribute>` 容器中，它的底层数据结构是一棵红黑树。因为在构造时，红黑树总是处于平衡状态，所以构建之后不需要调优。关于红黑树的细节，将在数据结构中讲解。`TreeMap<String, CoreDictionary.Attribute>` 加载的内容包括 4 部分：`String` 在这里是指词的字符串形式，`attribute.nature` 是词性信息（可以是多个），`attribute.frequency` 是不同词性下的词频信息，`attribute.totalFrequency` 是总词频。

`trie.build(map)` 一句调用了 `DoubleArrayTrie` 类的 `build` 方法，构建出一棵双数组 `trie` 树，代码如下。

```

/**
 * 方便地构造一棵双数组 trie 树
 * @param keyValuePair 升序键值对 map

```

```

    * @return 构造结果
    */
    public int build(TreeMap<String, V> keyValueMap) {
        assert keyValueMap != null;
        Set<Map.Entry<String, V>> entrySet = keyValueMap.entrySet(); //去重
        return build(entrySet); // 这里调用了真正的构建方法
    }

```

中等规模数据的存储和查询常用的数据结构是 `HashTable`，也称作散列表。Java 中的 `HashMap` 就是一种格式的散列表。散列表的基本原理是在表项的存储位置与其索引键（查询索引）之间建立一个对应函数关系，这个函数称为哈希函数，通过哈希函数计算得到每个索引键与表项的存储位置相对应的关系。这样，当输入查询键时，哈希表可以近似地得到随机访问的查询效率。这种数据结构因为对键的约束不高（不要求一定是整型数值），一直以来都是中等规模存储和查询的重要数据结构。

随着数据量的增大，对于大规模的数据存储（百万单位以上），哈希表的不足逐渐暴露出来。大规模数据对内存空间占用高，Hash 表因为需要哈希函数来计算地址映射，这必然会导致一些数据空间的损失（空桶），中小规模数据的情况下不明显，可以忽略不计。随着数据规模的增加，这种空间浪费会成倍地增加，导致有效存储空间的比例显著下降。

为此引入了一种新的词典结构 Trie 树，它也是搜索树的一种，其本质是一个确定有限状态自动机（DFA），每个节点代表自动机的一个状态，根据变量不同，进行状态转移，当到达结束状态或无法转移时，完成一次查询操作。考虑构造如下词表。

```

一举
一举一动
一举成名
一举成名天下知
万能
万能胶

```

这里提供一个节点的数据类型，然后通过这个节点来构造上述例子中的一棵 Trie 树：
`Node{code=0, depth=0, left=0, right=6}`，code 是字母的码表值，depth 是树深度，left 和 right 表示这个节点在词典中的索引范围。

如图 3.2 所示，黄色节点为空字符，代表从根节点到此节点的路径上的所有节点构成一个词。用 Trie 树搜索一个检索键的时间与键自身长度有关，最快的是 $O(1)$ ，即在第一层即可判断是否搜索到，最坏的情况是 $O(n)$ ， n 为 Trie 树的层数。

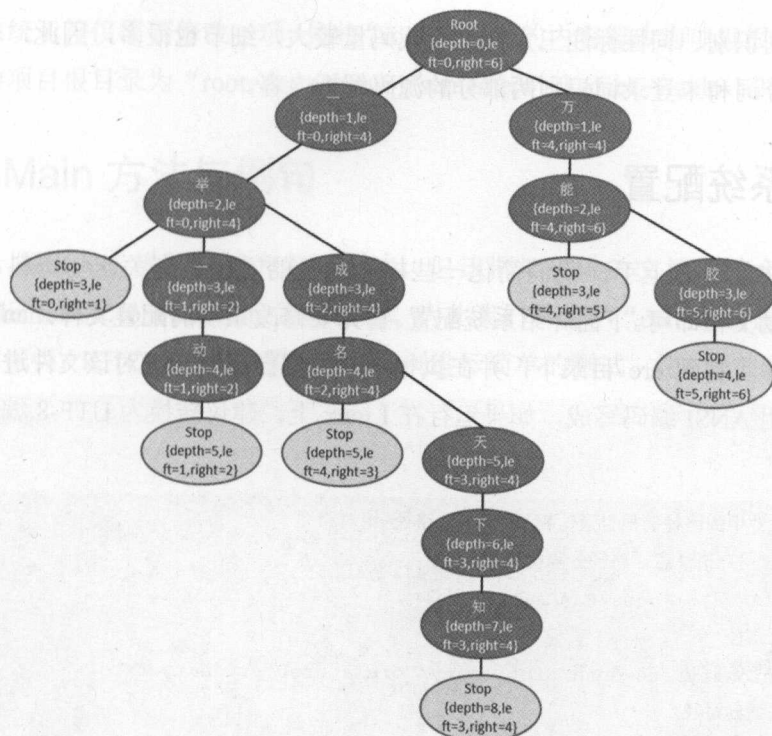


图 3.2 Tri 树结构

而双数组 Trie (Double-Array Trie) 结构就是将 Tire 树用两个整型数组来表示, 目的是为了设计一种 Trie 结构的压缩形式, 这个思想最初是由日本人 Jun-Ichi Aoe 于 1989 年提出的。仅用两个线性数组来表示 Trie 树, 该结构有效结合了数字搜索树 (Digital Search Tree) 检索时间高效的特点和链式表示的 Trie 空间结构紧凑的特点。其本质是一个确定有限状态自动机 (DFA)。DFA 的每个节点代表自动机的一个状态, 根据变量的不同, 进行状态转移, 当到达结束状态或无法转移时, 完成一次查询操作。在双数组所有键中包含的字符之间的联系都是通过简单的数学加法运算表示, 不仅提高了检索速度, 而且省去了链式结构中使用的大量指针, 节省了存储空间。

本文限于篇幅不做具体讲解, 有关双数组 Trie 树的更多细节内容, 读者可以参照源码了解。

3.3 算法部分源码解析

本节介绍了 HanLP 的源文件结构及部分核心源代码解析。由于整个项目包含中文分

词、未登录词识别、词性标注三大部分，代码量较大，细节也很多，因此，下文将把重点放在中文分词和未登录词识别两部分的源码解析内容。

3.3.1 系统配置

为了尽快使读者上手，我们简化一些构建项目的方法。有关这些资料，读者阅读 HanLP 的帮助文档即可。下面介绍系统配置。首先要修改系统的配置文件：`hanlp.properties`。然后将该文件复制到 `src` 目录下，并在执行之前根据自己的环境对该文件进行配置。该文件默认使用 ANSI 编码写成，如果运行在 Linux 上，建议转换为 UTF-8 编码。文件内容如下。

```
#本配置文件中的路径的根目录，根目录+其他路径=绝对路径
#Windows 用户请注意，路径分隔符统一使用/
root=D:/JavaProjects/HanLP/
#核心词典路径
CoreDictionaryPath=data/dictionary/CoreNatureDictionary.txt
#二元语法词典路径
BiGramDictionaryPath=data/dictionary/CoreNatureDictionary.ngram.txt
#停用词词典路径
CoreStopWordDictionaryPath=data/dictionary/stopwords.txt
#同义词词典路径
CoreSynonymDictionaryPath=data/dictionary/synonym/CoreSynonym.txt
#人名词典路径
PersonDictionaryPath=data/dictionary/person/nr.txt
#人名词典转移矩阵路径
PersonDictionaryTrPath=data/dictionary/person/nr.tr.txt
#繁简词典路径
TraditionalChineseDictionaryPath=data/dictionary/tc/TraditionalChinese.txt
#自定义词典路径，用;隔开多个自定义词典，空格开头表示在同一个目录，使用“文件名 词性”形式则表示这个词典的词性默认是该词性。优先级递减。
#另外 data/dictionary/custom/CustomDictionary.txt 是一个高质量的词库，请不要删除
CustomDictionaryPath=data/dictionary/custom/CustomDictionary.txt; 现代汉语补充词库.txt; 全国地名大全.txt ns; 人名词典.txt; 机构名词典.txt; 上海地名.txt ns;data/dictionary/person/nrf.txt nrf
#CRF 分词模型路径
CRFSegmentModelPath=data/model/segment/CRFSegmentModel.txt
#HMM 分词模型
HMMSegmentModelPath=data/model/segment/HMMSegmentModel.bin
#分词结果是否展示词性
ShowTermNature=true
```

如果系统运行仅需要修改一项,则将“root=”后面的内容修改为项目根目录。例如,改为本书的项目根目录为“root=/home/your_username/workspace/Hanlp-1.28/”。

3.3.2 Main 方法与例句

HanLP 源码并未提供测试的入口方法,但该框架提供的文档比较全,读者可从<http://hanlp.linrunsoft.com/doc.html> 页面查看。为此我们新建了“com.test.Hanlp;”包,并创建一个测试类:ICTCLASSeg。暂且以此为例进行简单的测试。主方法的源代码如下。

```
package com.test.Hanlp;

import com.hankcs.hanlp.seg.Segment;
import com.hankcs.hanlp.seg.NShort.NShortSegment;
import com.hankcs.hanlp.seg.Viterbi.ViterbiSegment;

public class ICTCLASSeg {
    public static void main(String[] args) {
        // 实例化 NShort 分词器,并启用地名、组织名词识别模块
        Segment nShortSegment = new NShortSegment().enableCustomDictionary(
            false).enablePlaceRecognize(true).enableOrganizationRecognize(true);
        // 输入例句
        String[] testCase = new String[]{
            "张强在铁岭对王小红说的确实在理。",
        };
        // 输出最短路径分词结果
        System.out.println("N-最短分词: " + nShortSegment.seg(testCase[0]) );
    }
}
```

参考教程的 NShortSegment 代码,简单实现了一个入口方法,可以看到执行结果如下。

```
N-最短分词: [乔布斯/nrf, 说/v, iphone/nx, 是/vshi, 最好/d, 用/p, 的/udel, 手机/n, 。/w]
```

这里 enableCustomDictionary()、enablePlaceRecognize()、enableOrganizationRecognize() 为命名实体识别的启用或关闭设置。这部分的详细代码在 NShortSegment 类中,比较简单,在此不做讲解。

3.3.3 句子切分

句子切分是中文分词的预处理阶段,这个节点的主要功能是对输入字符串按照分隔

符（全角分隔符、半角分隔符）来分隔句子。这个阶段使用的类是 `com.hankcs.hanlp.seg.Segment`，功能函数为 `seg()` 函数。

这些分隔符位于 `SentenceUtility` 类中，如下为参考源代码。

```
public static List<String> toSentenceList(char[] chars) {
    StringBuilder sb = new StringBuilder();
    List<String> sentences = new LinkedList<String>();
    for (int i = 0; i < chars.length; ++i) {
        if (sb.length() == 0 && (Character.isWhitespace(chars[i]) || chars[i]
        == ' ')) {
            continue;
        }
        sb.append(chars[i]);
        switch (chars[i]) {
            case '.':
                if (i < chars.length - 1 && chars[i + 1] > 128) {
                    insertIntoList(sb, sentences);
                    sb = new StringBuilder();
                }
                break;
            case '...': {
                if (i < chars.length - 1 && chars[i + 1] == '...') {
                    sb.append('...');
                    ++i;
                    insertIntoList(sb, sentences);
                    sb = new StringBuilder();
                }
            } break;
            case ' ':
            case ' ':
            case '?':
            case '。':
            case ',':
            case ',':
                insertIntoList(sb, sentences);
                sb = new StringBuilder();
                break;
            case ';':
            case ';':
                insertIntoList(sb, sentences);
                sb = new StringBuilder();
        }
    }
}
```



```

        break;
    case '!':
    case '! ':
        insertIntoList(sb, sentences);
        sb = new StringBuilder();
        break;
    case '?':
    case '? ':
        insertIntoList(sb, sentences);
        sb = new StringBuilder();
        break;
    case '\n':
    case '\r':
        insertIntoList(sb, sentences);
        sb = new StringBuilder();
        break;
    }
}

if (sb.length() > 0) {
    insertIntoList(sb, sentences);
}

return sentences;
}

```

这里所谓的句子，就是被一些标句符号分隔的字符串，这些标句符号包括：'...'、'!'、','、';'、':'、'!'、'!!'、'? '、'? '、'\n'、'\r'等（上述标点符号通过顿号来分隔）。句子切分的执行函数为 `toSentenceList`，该函数的逻辑很简单，就是将输入的一个字符串（可以理解为一个大的字符串）按照上述标句符号切分为字符串数组（这里是一个链表结构）。

如果输入的文本比较大，`Seg` 函数还支持多线程的调用来处理大型文本，源代码如下。

```

public List<Term> seg(String text) {
    char[] charArray = text.toCharArray(); //原子切分
    if (HanLP.Config.Normalization) {
        CharTable.normalization(charArray);
    }
    if (config.threadNumber > 1 && charArray.length > 10000) { // 小文本多线程
        // 没意义，反而变慢了
    }
}

```

```

        List<String> sentenceList = SentencesUtil.toSentenceList(charArray);
//将篇章切分为句子链表
        String[] sentenceArray = new String[sentenceList.size()];
        sentenceList.toArray(sentenceArray);
        List<Term>[] termListArray = new List[sentenceArray.length];
        final int per = sentenceArray.length / config.threadNumber;
        WorkThread[] threadArray = new WorkThread[config.threadNumber]; //创
建工作线程组
        for (int i = 0; i < config.threadNumber - 1; ++i) {
            int from = i * per;
            threadArray[i] = new WorkThread(sentenceArray, termListArray, from,
from + per); //创建线程
            threadArray[i].start(); //启动线程->run 方法
        }
        threadArray[config.threadNumber - 1] = new WorkThread(sentenceArray,
termListArray, (config.threadNumber - 1) * per, sentenceArray.length);
        threadArray[config.threadNumber - 1].start(); //启动线程->run 方法
        try {
            for (WorkThread thread : threadArray) {
                thread.join(); //合并执行完的线程
            }
        } catch (InterruptedException e) {
            logger.severe("线程同步异常: " + TextUtility.exceptionToString(e));
            return Collections.emptyList();
        }
        List<Term> termList = new LinkedList<Term>();
        if (config.offset || config.indexMode) {
            int sentenceOffset = 0; // 由于分割了句子,所以需要重新校正 offset
            for (int i = 0; i < sentenceArray.length; ++i) {
                for (Term term : termListArray[i]) {
                    term.offset += sentenceOffset;
                    termList.add(term);
                }
                sentenceOffset += sentenceArray[i].length();
            }
        } else {
            for (List<Term> list : termListArray) {
                termList.addAll(list);
            }
        }
        return termList;
    }
}

```

如下函数（run）就是 start 调用的多线程分词方法。

```
@Override
public void run() {
    for (int i = from; i < to; ++i) {
        termListArray[i] = segSentence(sentenceArray[i].toCharArray());
    }
}
```

分词后的结果都被保存在 termListArray 中。

3.3.4 分词流程

对输入的文本预处理完成（句子切分阶段结束）之后，即进入中文分词流程。NShortSegment 类完成了 NShort 中文分词算法的全过程。这个类继承自 WordBasedGenerativeModelSegment 类，而 WordBasedGenerativeModelSegment 类又继承自 Segment 类。

HanLP 不仅提供了 NShort 分词算法，还提供了基于 HMM、CRF、Dijkstra 等的多种切分和标注方法。所有这些分词方法都继承自 Segment 类，而 NShort 和 HMM 在概率图模型中均属于产生式模型。因此，这两个类继承自 WordBasedGenerativeModelSegment 类。通过如此复杂的继承关系，系统将若干个分词算法形成一个整体。

在 CResult::Processing 函数中，这里主要分析中文分词的完整流程。其流程如图 3.3 所示。

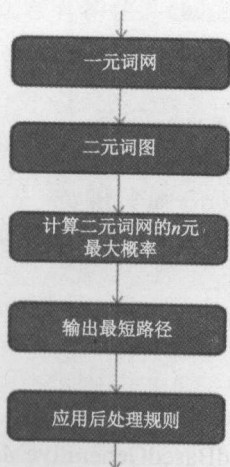


图 3.3 中文分词的流程

如图 3.3 所示,分词的过程包含 4 个大的步骤,分别为:一元词网、二元词图、输出最短路径、应用后处理规则。

(1) 一元词网。根据输入的字符串,查询核心词典,将原子分词的结果与词典中的词进行最大匹配。匹配的结果包括词字符串、词性、词频等信息形成一个二维数组,并将此数组存入 WordNet 二维数组中。

(2) 二元词图。用一元分词的结果(二维数组)查询二元词典,与二元词典进行最大匹配。匹配的结果为一个二维数组,并将此结果存入新的 Graph 中,形成一个词图。

(3) 输出最短路径。将查出的每个结果按平滑算法计算一元分词和二元分词的词频数得到词网中每个节点的权值(概率的倒数),应用 NShort 算法累加词网中每个节点构成的所有路径,权值最小(概率最大)的那条路径对应的词网节点就是初分的结果。

(4) 应用后处理规则。对粗分结果应用规则,识别特殊组合的名词性结构。

NShortSegment 类的 segSentence()是控制分词流程的主函数。如下是经过注释的粗分词阶段的函数代码片段。

```
@Override
public List<Term> segSentence(char[] sentence)
{
    WordNet wordNetOptimum = new WordNet(sentence); //最优词网
    WordNet wordNetAll = new WordNet(sentence); //一元词网
    // 粗分主函数
    List<List<Vertex>> coarseResult = BiSegment(sentence, 2, wordNetOptimum,
wordNetAll);
    boolean NERexists = false;
    for (List<Vertex> vertexList : coarseResult)
    {
        if (HanLP.Config.DEBUG)
        {
            System.out.println("粗分结果" + convert(vertexList, false));
        }
    }
    .....
```

3.3.5 一元词网

构建一元词网的过程是由 WordBasedGenerativeModelSegment 类的 GenerateWordNet 方法完成的。部分源代码如下。

```

/**
 * 二元语言模型分词
 * @param sSentence 待分词的字符数组
 * @param nKind 返回的结果数
 * @param wordNetOptimum 最优词网
 * @param wordNetAll 全部词网
 * @return 一系列粗分结果
 */
public List<List<Vertex>> BiSegment(char[] sSentence, int nKind, WordNet
wordNetOptimum, WordNet wordNetAll)
{
    List<List<Vertex>> coarseResult = new LinkedList<List<Vertex>>(); //粗分结
果链表

    ////////////生成词网//////////
    GenerateWordNet(wordNetAll);

    if (HanLP.Config.DEBUG)
        System.out.printf("打印词网: \n" + wordNetAll);
}

```

构建一元词网的流程分为如下三个部分。

1. 最大匹配查询词典

最大匹配是查找词典的一种方法。我们希望从词典中查找某个字符串成词的情况，如字符串“中华人民共和国”，从“中”开始查询词典，找到两个词“中”、“中华”；然后从“华”开始再查询词典，找到一个词“华人”；依次向下直至找到所有的词为止。HanLP 完成最大匹配的过程比较有特色。GenerateWordNet 中对该函数的调用代码如下。

```

.....
// 核心词典查询——一种最大匹配的特色实现

DoubleArrayTrie<CoreDictionary.Attribute>.Searcher searcher = CoreDictionary.
trie.getSearcher(charArray, 0); //构建搜索器
while (searcher.next()) //不断向下一步搜索
{
    //在词网中加入搜索到的节点
    // Vertex: 词内容, value: 词性、词频对, index: 该词在一元词典中的序号
    wordNetStorage.add(searcher.begin + 1, new Vertex(new String(charArray,
searcher.begin, searcher.length), searcher.value, searcher.index));
}
.....

```

词典的数据结构是一个双数组 Tri 树。因此，首先构造一个 Tri 树的查询器，从起始位置依次向后遍历整个字符数组，返回了词的字符串内容、词性、词频、词典中的位置等信息。有时候可以查询到一个词汇有多个词性，每个词性都有不同的词频。这样，查询结果需要记录每个词性和词频信息，并汇总所有的词频计算出总的词频数。

最后，使用这些信息构造一个图的顶点 Vertex，将其保存在 WordNet 一元词网中。如下为节点主构造方法的代码，其他构造都是它的一个变种。

```
public Vertex(String word, String realWord, CoreDictionary.Attribute attribute,
int wordID)
{
    if (attribute == null) attribute = new CoreDictionary.Attribute(Nature.n,
1); // 安全起见
    this.wordID = wordID;
    this.attribute = attribute;
    //将原词变为等效词串
    if (word == null) word = compileRealWord(realWord, attribute);
    assert realWord.length() > 0 : "构造空白节点会导致死循环! ";
    this.word = word;
    this.realWord = realWord;
}
```

这里有一个重要的函数 compileRealWord。该函数生成了 realWord。生成的类型与词性对照如表 3.6 所示。

表 3.6 生成的类型与词性对照

序 号	标 识	等效字符串	词 性	含 义
1	TAG_PLACE	"未##地"	ns、nsf	地名
2	TAG_PEOPLE	"未##人"	nr、nr1、nr2、nrf、nrj	人名
3	TAG_PROPER	"未##专"	nx	专有名词
4	TAG_GROUP	"未##团"	nt、ntc、ntcf、ntcb、ntch、 nto、ntu、nts、nth、nit	组织机构名
5	TAG_NUMBER	"未##数"	m、mq	数字类型
6	TAG_CLUSTER	"未##串"	x	字符串类型
7	TAG_TIME	"未##时"	t	时间词

2. 原子切分和确定类型

一元词网初步构建后，GenerateWordNet 剩下的过程是对一元词网中的每个词进行原子切分，并确定其类型。


```

.....
// 原子分词, 保证图连通
LinkedList<Vertex>[] vertexes = wordNetStorage.getVertexes();
for (int i = 1; i < vertexes.length; )
{
    if (vertexes[i].isEmpty())
    {
        int j = i + 1;
        for (; j < vertexes.length - 1; ++j)
        {
            if (!vertexes[j].isEmpty()) break;
        }
        //快速原子切分, charArray: 字符数组, i - 1: 起始下标, j - 1: 终止下标
        wordNetStorage.add(i, quickAtomSegment(charArray, i - 1, j - 1));
        i = j;
    }
    else i += vertexes[i].getLast().realWord.length();
}
}

```

生成原子词的方法为 `quickAtomSegment`, 函数调用了 `CharType` 类, 为每个从词典中查询出的词给出原子词的类型。

系统使用 UTF-8 编码方式。因为 UTF-8 下的汉字编码和全角符号编码没有规律性, 不能用一个简单的程序实现, 所以所有汉字、全角、半角符号类型的 `CharType` 都被保存在一个表中, 文件路径为 “data/dictionary/other/CharType.dat.yes”。在 `CharType` 类初始化时该文件加载到内存。解析该文件的静态代码段如下。

```

static
{
    type = new byte[65536];
    logger.info("字符类型对应表开始加载 " + HanLP.Config.CharTypePath);
    long start = System.currentTimeMillis();
    ByteArray byteArray = ByteArray.createByteArray(HanLP.Config.CharTypePath);
    if (byteArray == null) {
        System.err.println("字符类型对应表加载失败: " + HanLP.Config.CharTypePath);
        System.exit(-1);
    } else{
        while (byteArray.hasMore()) {
            int b = byteArray.nextChar();
            int e = byteArray.nextChar();
            byte t = byteArray.nextByte();

```

```
        for (int i = b; i <= e; ++i) {
            type[i] = t;
        }
    }
    logger.info("字符类型对应表加载成功, 耗时" + (System.currentTimeMillis() -
start) + " ms");
}
}
```

quickAtomSegment 函数在查询每个字符的 CharType 时,都要调用内存中的这个表,生成每个字符的原子词类型。一些非汉字的原子词类型需要在 quickAtomSegment 进行合并处理,处理的结果类型定义在 CharType 类中。综合所有非汉字的 CharType 类型将其总结为表 3.7。

表 3.7 CharType类型标签

序 号	标 识	等效字符串	词 性	含 义
5	CT_SINGLE	"未##串"	nx	单字节字符
6	CT_DELIMITER	查表值	w	分隔符
7	CT_CHINESE	查表值	原词性	汉字字符类型
8	CT_LETTER	"未##串"	nx	英文字符类型
9	CT_NUM	"未##数"	m	数字字符类型
17	CT_OTHER	—	—	其他类型字符

生成查询出每个顶点的 CharType 编码之后,对其中汉字类型不做任何改变,但合并浮点类型的数字(“3,,1,4”合并为 3.14);合并 ASCII 的字符为一个原子词(将“i,p,h,o,n,e”合并为一个词 iphone);合并数字为一个原子类型(将“1,0,0”合并为 100)。经过这些修改之后,原来的句子发生了一些变化,这个过程称为原子切分。本节最后将显示原子切分的结果。

最后,将生成的原子词序列放入一元词网,生成新的节点,代码如下。

```
/**
 * 添加顶点,由原子分词顶点添加
 */
public void add(int line, List<AtomNode> atomSegment)
{
    int offset = 0;
    for (AtomNode atomNode : atomSegment)//Init the cost array
    {
        String sWord = atomNode.sWord;//init the word
```

```

        Nature nature = Nature.n;
        switch (atomNode.nPOS)
        {
            case Predefine.CT_CHINESE:
                break;
            case Predefine.CT_INDEX:
            case Predefine.CT_NUM:
                nature = Nature.m;
                sWord = "未##数";
                break;
            case Predefine.CT_DELIMITER:
                nature = Nature.w;
                break;
            case Predefine.CT_LETTER:
                nature = Nature.nx;
                sWord = "未##串";
                break;
            case Predefine.CT_SINGLE://12021-2129-3121
                nature = Nature.nx;
                sWord = "未##串";
                break;
            default:
                break;
        }
        add(line + offset, new Vertex(sWord, atomNode.sWord, new CoreDictionary.
Attribute(nature, 1)));
        offset += atomNode.sWord.length();
    }
}

```

该函数处理各种 CharType 类型的原子词，汉字不变，其他类型都给出了新的词性及 CharType 类型的字符串标识。

3. 构建一元词网

对句子完成了原子切分和类型合并，下一步需要利用这个切分结果构建出一元词网。为了便于读者的深刻理解，我们回到 NShortSegment 类的 BiSegment 主函数的代码片段，查看如下执行的结果。

```

//一元分词
/**
 * 二元语言模型分词

```



```

* @param sSentence 待分词的字符数组
* @param nKind      需要几个结果
* @param wordNetOptimum 最优词网
* @param wordNetAll 全部词网
* @return 一系列粗分结果
*/
public List<List<Vertex>> BiSegment(char[] sSentence, int nKind, WordNet
wordNetOptimum, WordNet wordNetAll)
{
    List<List<Vertex>> coarseResult = new LinkedList<List<Vertex>>(); //粗分节点
    //生成词网
    GenerateWordNet(wordNetAll);
    if (HanLP.Config.DEBUG)
        System.out.printf("打印词网: \n" + wordNetAll);
    ...
}

```

为了更清晰地观察，这里修改了 `WordNet.toString()` 方法，以便给出这个阶段的输出结果的更多细节。

```

@Override
public String toString()
{
    StringBuilder sb = new StringBuilder();
    int line = 0;
    for (List<Vertex> vertexList : vertexes)
    {
        sb.append(String.valueOf(line++) + ':' );
        //sb.append(String.valueOf(line++) + ':' + vertexList.toString());
        append("\n");
        for (Vertex vertex:vertexList){
            sb.append "[" + vertex.getRealWord() + "\t" + vertex.word + " ]";
        }
        sb.append('\n');
    }
    return sb.toString();
}

```

例句 1 测试代码输出结果如下。

```

打印词网:
0:[ 始##始]
1:[石    石]
2:[国    国]

```

```

3:[祥    祥]
4:[会    会][会见 会见]
5:[见    见]
6:[乔    乔][乔布 未##人][乔布斯 未##人]
7:[布    布][布斯 未##人]
8:[斯    斯]
9:[说    说]
10:[iphone 未##串]
11:
12:
13:
14:
15:
16:[是    是]
17:[最    最][最好 最好]
18:[好    好]
19:[用    用]
20:[的    的]
21:[手    手][手机 手机]
22:[机    机]
23:[。    。]
24:[      未##末]

```

以行“6:[乔 乔][乔布 未##人][乔布斯 未##人]”为例。输出表的第一列为序号(6:)，即输入的句子字符串转换为字符数组的索引位置。第二列为一个或多个“[]”，如[乔 乔]、[乔布 未##人]、[乔布斯 未##人]。它们是查询词典后的结果，存储为一元词网的子链节点。“[]”内的文字被空格分隔为两部分：一部分为节点的 `realword` 属性；另一部分为 `word` 属性。`word` 的属性来源于 `CharType` 表的查询结果(具体见上文)，可能是汉字本身，也可能是某个 `CharType` 类型的字符串标识形式。除此之外，顶点还保存了查询出的词汇的词频和词性信息。

第一个顶点和最后一个顶点为标识句首和句尾的顶点，`word` 属性为“始##始”、“末##末”，没有 `realword` 属性，因为 `realword` 属性值来源于切分后的原子词。

表中 11~15 行为空行，因为这里的字符都被合并到第 10 行，这是由原子切分所产生的结果。

3.3.6 二元词图

通过一元词网得到一个二维数组(严格意义上是一张广义表)。该数组包含所有的原

子词，以及相应的词频和词性信息。二元分词是将这些原子词和二元词典词按顺序排列组合，产生词与词相关联的词图，并计算这个关联词图中每个二元关联词的词频概率，通过这个过程消除分词中的歧义问题，并为后面的 NShort 最短路径计算做准备。

```

//////////生成词图//////////
Graph graph = GenerateBiGraph(wordNetAll);
if (HanLP.Config.DEBUG)
{
    System.out.printf("打印词图: %s\n", graph.printByTo());
}

```

如果把一元的分词结果当作一个节点，那么二元关联词即可看作节点之间的一条连线，也就是边，而含有节点和边的数据结构即可被看作一张图。下面来详细了解如何生成一张二元词图的过程。

词图的数据结构包含两个类，一个是 Graph 类，代码如下。

```

public class Graph
{
    public Vertex[] vertexes; // 顶点
    public List<EdgeFrom>[] edgesTo; // 边，到达下标 i
    .....
}

```

仅包含顶点列表和顶点的到达边列表。

一个是 Edge 类及其子类 EdgeFrom 类。Edge 类的代码如下。

```

/**
 * 基础边，不允许构造
 */
public class Edge
{
    public double weight; // 花费
    String name; // 节点名字，调试用
    .....
}

```

EdgeFrom 类的代码如下。

```

/**
 * 记录了起点的边
 */
public class EdgeFrom extends Edge
{
    public int from; // 起点的索引
    .....
}

```


这是标准的图类型的数据结构，并不复杂。

二元分词的算法是通过 `toGraph` 这个函数完成的。其源代码如下。

```
/**
 * 词网转词图
 * @return 词图
 */
public Graph toGraph()
{
    Graph graph = new Graph(getVertexesLineFirst());
    for (int row = 0; row < vertexes.length - 1; ++row)
    {
        List<Vertex> vertexListFrom = vertexes[row];
        for (Vertex from : vertexListFrom) {
            assert from.realWord.length() > 0 : "空节点会导致死循环! ";
            int toIndex = row + from.realWord.length(); //计算边连接节点的下标
            for (Vertex to : vertexes[toIndex]) //可能连接多个节点
            {
                //构建词图的边，并计算权重
                graph.connect(from.index, to.index, MathTools.calculateWeight
(from, to));
            }
        }
    }
    return graph;
}
```

该函数从词网首行顶点开始创建词图。主循环遍历顶点列表，计算连接每个顶点的各条边的到达位置索引，然后使用 `graph.connect` 方法创建出对应的边，以及计算出该边的权重信息。权重信息的计算公式原理来自贝叶斯公式。公式的计算参见《基于 N-最短路径方法的中文词语粗分模型》的一元粗分模型的求解与实现一节。

计算权重信息的 `calculateWeight` 方法如下。

```
/**
 * 从一个词到另一个词的词的花费
 * @param from 前面的词
 * @param to 后面的词
 * @return 分数
 */
public static double calculateWeight(Vertex from, Vertex to) {
    int frequency = from.getAttribute().totalFrequency;
    if (frequency == 0) {
```

```

        frequency = 1; // 防止发生除零错误
    }
    int nTwoWordsFreq = CoreBiGramTableDictionary.getBiFrequency(from.wordID,
to.wordID);
    double value = -Math.log(dSmoothingPara * frequency / (MAX_FREQUENCY) + (1
- dSmoothingPara) * ((1 - dTemp) * nTwoWordsFreq / frequency + dTemp));
    if (value < 0.0) {
        value = -value;
    }
    return value;
}

```

其中, `frequency` 是边的起始顶点的词频; `nTwoWordsFreq` 是二元节点的词频; `value` 为最终的计算结果。这个结果作为二元词图的边的权重, 参与到最终的 NShort 路径计算。`value` 的计算需要参考如下几个参数。

```

public static final int MAX_FREQUENCY = 25146057; // 语料库总词频 25146057
public static final double dTemp = (double) 1 / MAX_FREQUENCY + 0.00001;
// Smoothing 平滑因子
public static final double dSmoothingPara = 0.1; //平滑参数

```

构建二元词图边的程序如下。

```

/**
 * 连接两个节点
 * @param from 起点
 * @param to 终点
 * @param weight 权重
 */
public void connect(int from, int to, double weight)
{
    edgesTo[to].add(new EdgeFrom(from, weight, vertexes[from].word + '@' +
vertexes[to].word));
}

```

开启调试功能, 运行程序, 可以看到二元分词过程中的输出如下。

```

打印词图: =====按终点打印=====
to: 1, from: 0, weight:04.60, word:始##始@石
to: 2, from: 1, weight:11.36, word:石@国
to: 3, from: 2, weight:10.85, word:国@祥
to: 4, from: 3, weight:11.57, word:祥@会
to: 5, from: 3, weight:11.57, word:祥@会见

```

```

to: 6, from: 4, weight:08.81, word:会@见
to: 7, from: 5, weight:11.40, word:会见@乔
to: 7, from: 6, weight:10.91, word:见@乔
to: 8, from: 5, weight:02.92, word:会见@未##人
to: 8, from: 6, weight:10.91, word:见@未##人
to: 9, from: 5, weight:02.92, word:会见@未##人
to: 9, from: 6, weight:10.91, word:见@未##人
to: 10, from: 7, weight:11.60, word:乔@布
to: 11, from: 7, weight:11.60, word:乔@未##人
to: 12, from: 8, weight:11.61, word:未##人@斯
to: 12, from: 10, weight:11.52, word:布@斯
to: 13, from: 9, weight:06.60, word:未##人@说
to: 13, from: 11, weight:09.20, word:未##人@说
to: 13, from: 12, weight:11.38, word:斯@说
to: 14, from: 13, weight:04.02, word:说@未##串
to: 15, from: 14, weight:06.80, word:未##串@是
to: 16, from: 15, weight:04.67, word:是@最
to: 17, from: 15, weight:06.06, word:是@最好
to: 18, from: 16, weight:09.49, word:最好@
to: 19, from: 17, weight:03.72, word:最好@用
to: 19, from: 18, weight:05.48, word:好@用
to: 20, from: 19, weight:02.71, word:用@的
to: 21, from: 20, weight:05.41, word:的@手
to: 22, from: 20, weight:05.32, word:的@手机
to: 23, from: 21, weight:10.91, word:手@机
to: 24, from: 22, weight:02.81, word:手机@
to: 24, from: 23, weight:02.25, word:机@
to: 25, from: 24, weight:11.61, word:@未##末

```

word 是拼接后的二元关联词，中间用@进行连接；weight 是平滑计算后的词频；from 是位于@前面词的行下标（起始节点）；to 是位于@后面词的列下标（终止节点）。

一元分词表中的任何一个词都被看作一个节点，两个相邻行的任意节点之间都存在着一种共现关系。二元关联词就是描述这一共现关系的词汇组合，二元关联词的概率能反映在语料中两个一元词汇之间搭配的频繁程度，而此共现关系的定量化就是要计算二元关联词在语料中的概率，由此判断出哪个一元词汇更有可能在此句中是正确的中文词，从而消除前文所讲的两种歧义现象。

图 3.4 非常清晰地描述了二元词图的实际功能。算法首先按行从上到下，将本行的每个元素与位于下一行的每个元素建立一个关联词，这种关联关系就构成了图中的一条边。例如，“祥”下面有两个节点：一个是“会”；另一个是“会见”。于是建立了两条边

“祥@会”和“祥@会见”，如果上层节点有多个，如“斯”、“乔布斯”，那么下层要分别连接上层的每个节点：“乔布”→“乔布@斯”，“布”→“布@斯”；“会见”→“会见@乔布斯”，“见”→“见@乔布斯”。之后，根据生成的结果从二元词典中查询并计算二元词图的边的概率。最后，将生成的结果保存在一个新的二元链表中。

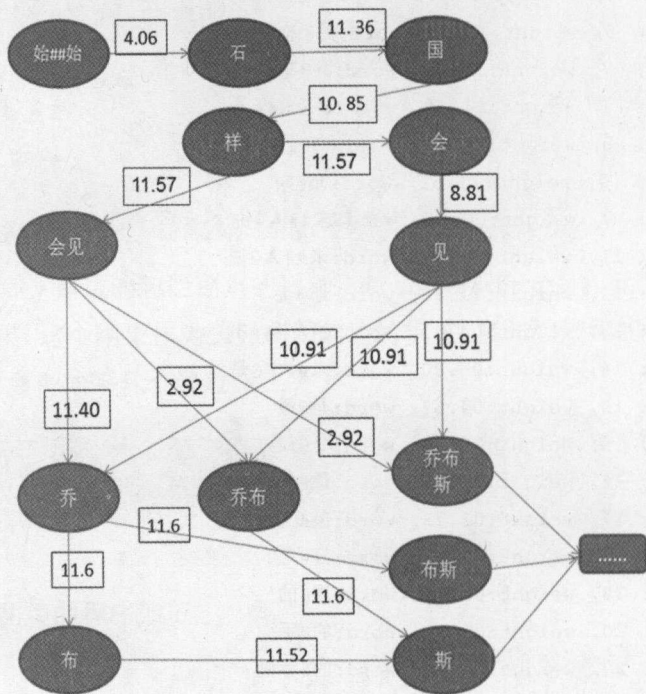


图 3.4 二元词图（部分）

3.3.7 NShort 算法原理

下面，到了最关键的时刻，计算 NShort 最短路径的算法流程。NShort 最短路径的调用程序如下。

```
...  
//////////N-最短路径//////////  
NShortPath nShortPath = new NShortPath(graph, nKind); // 根据词图生成 NShort  
路径  
List<int[]> spResult = nShortPath.getNPaths(nKind * 2); // 获得最短路径  
if (spResult.size() == 0) {  
    throw new RuntimeException(nKind + "-最短路径求解失败，请检查上述词网是否存在负圈或悬孤节点");  
}  
...
```

最短路径的计算分为如下两个步骤。第一个步骤是从词图中构建出节点和权重排序数组，第二个步骤是从词图中过滤掉排序数组中权重较高的路径，并根据权重较低的路径生成最终的分词结果。

NShortPath 的 calculate 方法完成了上述第一个步骤，如下代码为其上半部分。

```
/**
 * 计算出所有节点上可能的路径，以及各条路径的累计权重
 * 为路径数据提供数据准备
 * @param inGraph 输入图
 * @param nValueKind 前 N 个结果，粗分阶段返回前两个结果。
 */
private void calculate(Graph inGraph, int nValueKind) {
    initNShortPath(inGraph, nValueKind); //构建节点和权重队列——堆
    QueueElement tmpElement;
    CQueue queWork = new CQueue();
    double eWeight;
    for (int nCurNode = 1; nCurNode < vertexCount; ++nCurNode) {
        // 将所有到当前节点 (nCurNode) 可能到达的边根据 eWeight 排序, 然后压入队列
        enqueueCurNodeEdges(queWork, nCurNode);
        System.out.print(queWork.toString()); // 临时变量保存了到当前节点的所有可
        能的边及其累计权重
        System.out.println("-----curNode:"+nCurNode+"-----\n");
    }
    ...
}
```

输出结果如表 3.8 所示。

表 3.8 累计路径表（部分）

终止 节点	起始节点, 边权重	终止 节点	起始节点, 权重
to:1	from:0 weight:4.6024452831351015	to:13	from:9 weight:47.900136643769365
			from:12 weight:64.29618617352028
			from:9 weight:64.69978163492513
			from:11 weight:70.58489622392035
			from:11 weight:78.89890465912715
to:2	from:1 weight:15.95834780869178	to:14	from:12 weight:81.09583116467604
			from:13 weight:51.9238658522582
to:3	from:2 weight:26.812938442096577	to:15	from:13 weight:68.31991538200913
			from:14
			weight:58.726250585808735
			from:14 weight:75.12230011555967

续表

终止节点	起始节点, 边权重	终止节点	起始节点, 权重
to:4	from:3 weight:38.38123868491938	to:16	from:15 weight:63.40040559925871 from:15 weight:79.79645512900964
to:5	from:3 weight:38.38123868491938	to:17	from:15 weight:64.78416669522657 from:15 weight:81.1802162249775
to:6	from:4 weight:47.19009705655819	to:18	from:16 weight:72.88800497247385 from:16 weight:89.28405450222479
to:7	from:5 weight:49.784824884439445 from:6 weight:58.098833319646246	to:19	from:17 weight:68.50376658858679 from:18 weight:78.36726655289716 from:17 weight:84.89981611833771 from:18 weight:94.7633160826481
to:8	from:5 weight:41.29918832849048 from:6 weight:58.098833319646246	to:20	from:19 weight:71.21175447914247 from:19 weight:81.07525444345285
to:9	from:5 weight:41.29918832849048 from:6 weight:58.098833319646246	to:21	from:20 weight:76.62603550377729 from:20 weight:86.48953546808767
to:10	from:7 weight:61.3812582347129 from:7 weight:69.6952666699197	to:22	from:20 weight:76.53552508147455 from:20 weight:86.39902504578492
to:11	from:7 weight:61.3812582347129 from:7 weight:69.6952666699197	to:23	from:21 weight:87.54087580023351 from:21 weight:97.40437576454389
to:12	from:8 weight:52.91306541302703 from:8 weight:69.71271040418279 from:10 weight:72.90516419305263 from:10 weight:81.21917262825943	to:24	from:22 weight:79.34102376343522 from:22 weight:89.2045237277456 from:23 weight:89.78644517666935 from:23 weight:99.64994514097972
		to:25	from:24 weight:90.95490084797176 from:24 weight:100.81840081228214

根据路径选择 2，过滤掉无效的权重，保留有效的权重，如下代码为其下半部分。

```
...
// 初始化当前节点所有边的 eWeight 值
for (int i = 0; i < N; ++i) weightArray[nCurNode - 1][i] =
Double.MAX_VALUE;

tmpElement = queWork.dequeue(); // 将 queWork 中的内容装入 fromArray
if (tmpElement != null) {
    for (int i = 0; i < N; ++i) {
        eWeight = tmpElement.weight;
        weightArray[nCurNode - 1][i] = eWeight;
```



```
do {  
    fromArray[nCurNode - 1][i].enqueue(new QueueElement(tmpElement.  
from, tmpElement.index, 0));  
    tmpElement = queWork.dequeue();  
    if (tmpElement == null) {  
        i = N;  
        break;  
    }  
} while (tmpElement.weight == eWeight);  
}  
}  
}
```

因为处于粗分阶段，对照源码，其中 nKind 值为 2，即返回前两个结果，过滤掉权重较大的结果，则输出结果如表 3.9 所示。

表 3.9 优化累计路径表（部分）

终止节点	起始节点，边权重	终止节点	起始节点，权重
to:1	from:0 weight:4.6024452831351015	to:13	from:9 weight:47.900136643769365 from:12 weight:64.29618617352028
to:2	from:1 weight:15.95834780869178	to:14	from:13 weight:51.9238658522582 from:13 weight:68.31991538200913
to:3	from:2 weight:26.812938442096577	to:15	from:14 weight:58.726250585808735 from:14 weight:75.12230011555967
to:4	from:3 weight:38.38123868491938	to:16	from:15 weight:63.40040559925871 from:15 weight:79.79645512900964
to:5	from:3 weight:38.38123868491938	to:17	from:15 weight:64.78416669522657 from:15 weight:81.1802162249775
to:6	from:4 weight:47.19009705655819	to:18	from:16 weight:72.88800497247385 from:16 weight:89.28405450222479
to:7	from:5 weight:49.784824884439445 from:6 weight:58.098833319646246	to:19	from:17 weight:68.50376658858679 from:18 weight:78.36726655289716
to:8	from:5 weight:41.29918832849048 from:6 weight:58.098833319646246	to:20	from:19 weight:71.21175447914247 from:19 weight:81.07525444345285
to:9	from:5 weight:41.29918832849048 from:6 weight:58.098833319646246	to:21	from:20 weight:76.62603550377729 from:20 weight:86.48953546808767

续表

终止节点	起始节点, 边权重	终止节点	起始节点, 权重
to:10	from:7 weight:61.3812582347129 from:7 weight:69.6952666699197	to:22	from:20 weight:76.53552508147455 from:20 weight:86.39902504578492
to:11	from:7 weight:61.3812582347129 from:7 weight:69.6952666699197	to:23	from:21 weight:87.54087580023351 from:21 weight:97.40437576454389
to:12	from:8 weight:52.91306541302703 from:8 weight:69.71271040418279	to:24	from:22 weight:79.34102376343522 from:22 weight:89.2045237277456
		to:25	from:24 weight:90.95490084797176 from:24 weight:100.81840081228214

将返回的节点和权重列表拼成最短路径并输出，结果如下。

```
/**
 * 获取前 index+1 短的路径
 * @param index index = 0 : 最短的路径; index = 1 : 次短的路径, 依次类推。index <=
this.N
 * @return
 */
public List<int[]> getPaths(int index) {
    assert (index <= N && index >= 0);
    Stack<PathNode> stack = new Stack<PathNode>();
    int curNode = vertexCount - 1, curIndex = index;
    QueueElement element;
    PathNode node;
    int[] aPath;
    List<int[]> result = new ArrayList<int[]>();
    element = fromArray[curNode - 1][curIndex].GetFirst();
    while (element != null) {
        // ----- 通过压栈得到路径 -----
        stack.push(new PathNode(curNode, curIndex));
        stack.push(new PathNode(element.from, element.index));
        curNode = element.from;
        while (curNode != 0) {
            element = fromArray[element.from - 1][element.index].GetFirst();
            System.out.println(element.from + " " + element.index); //测试检查最
            stack.push(new PathNode(element.from, element.index));
            curNode = element.from;
        }
    }
}
```

优路径

```

// ----- 输出路径 -----
PathNode[] nArray = new PathNode[stack.size()];
for (int i = 0; i < stack.size(); ++i) {
    nArray[i] = stack.get(stack.size() - i - 1);
}
aPath = new int[nArray.length];
for (int i = 0; i < aPath.length; i++)    aPath[i] = nArray[i].from;
result.add(aPath);
// ----- 出栈以检查是否还有其他路径 -----
do {
    node = stack.pop();
    curNode = node.from;
    curIndex = node.index;

    } while (curNode < 1 || (stack.size() != 0 && !fromArray[curNode - 1][curIndex].CanGetNext()));
    element = fromArray[curNode - 1][curIndex].GetNext();
}
return result;
}

```

输出结果如下。

最优路径 (1): 22->20->19->17->15->14->13->9->5->3->2->1->0。

最优路径 (2): 22->20->19->18->16->15->14->13->9->5->3->2->1->0。

选取最优的计算结果, 即最优路径 1), 如图 3.5 所示, 图解的部分最优路径 1), 标有数字的边就是计算后的最短路径。

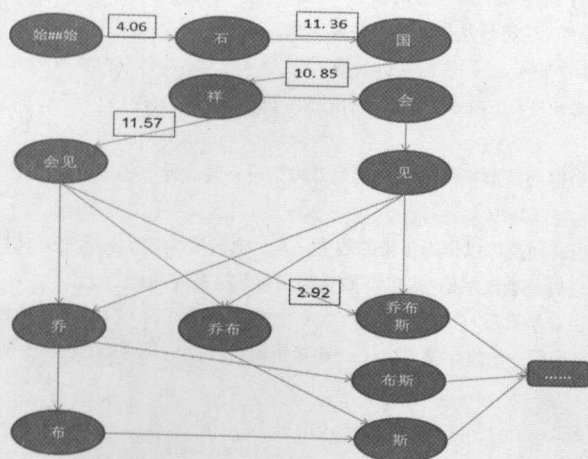


图 3.5 NShort 最短路径 (部分)

3.3.8 后处理规则集

HanLP 重写了 ICTCLAS 版的 `GenerateWord` 方法, 使该函数的结构变得更加清晰, 可以算作 ICTCLAS 中行数最多的函数。该函数主要编写了分词必需的后处理规则集。函数代码量虽然很大, 但是逻辑并不复杂。该函数的简化的流程如下。

```

//////////日期、数字合并策略
for (int[] path : spResult) {
    List<Vertex> vertexes = graph.parsePath(path); // 将路径解析为节点数组
    GenerateWord(vertexes, wordNetOptimum); // 应用后处理规则
    coarseResult.add(vertexes); // 输出粗分结果
}
return coarseResult;
}

```

`fixResultByRule` 函数是 ICTCLAS 中 `Generate` 函数的简化版, 可以很清晰地看到函数包含了大量的规则。下面具体看看都有哪些规则。通过对源代码的分析, 从函数列出的规则中抽取出几个有代表性的, 并做如下简单的说明。

```

// 通过规则修正一些结果
protected static void fixResultByRule(List<Vertex> linkedArray) {

    //Merge all separate continue num into one number
    mergeContinueNumIntoOne(linkedArray);

    //The delimiter "--"
    ChangeDelimiterPOS(linkedArray);

    //如果前一个词是数字, 当前词以“-”或“-”开始, 并且不止这一个字符,
    //那么将此“-”符号从当前词中分离出来。
    //例如 “3 / -4 / 月” 需要拆分成 “3 / - / 4 / 月”
    SplitMiddleSlashFromDigitalWords(linkedArray);

    //1. 如果当前词是数字, 下一个词是“月、日、时、分、秒、月份”中的一个, 则合并, 且当前
    词的词性是时间。
    //2. 如果当前词是可以作为年份的数字, 下一个词是“年”, 则合并, 词性为时间, 否则为数字。
    //3. 如果最后一个汉字是“点”, 则认为当前数字是时间。
    //4. 如果当前串最后一个汉字不是“:”、“.”和半角的“'”、“/”, 那么是数。
    //5. 当前串最后一个汉字是“:”、“.”和半角的“'”、“/”, 且长度大于1, 那么去掉最后一个字符。
    例如“1.”

    CheckDateElements(linkedArray);
}

```

粗分结果如下。

粗分结果[石/ng, 国/n, 祥/ag, 会见/v, 乔布斯/nrf, 说/v, iphone/nx, 是/vshi, 最好/d, 用/p, 的/udel, 手机/n, 。/w]

3.3.9 命名实体识别

初分阶段结束,系统提供了一系列的命名实体识别程序来识别不同类型的命名实体,包括中国人名、日本人名、译名、地名和组织机构名。**HanLP** 的命名实体识别使用标准的 HMM 模型和 Viterbi 算法。有关算法的细节,将在第 4 章详细讲解。本节以人名识别模块为例,给出整个模块的基本执行过程。

```
.....
// 实体命名识别

    if (config.ner) {
        wordNetOptimum.addAll(vertexList);
        int preSize = wordNetOptimum.size();
        if (config.nameRecognize) {
            PersonRecognition.Recognition(vertexList, wordNetOptimum, wordNetAll);
        }
        if (config.translatedNameRecognize) {
            TranslatedPersonRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
        }
        if (config.japaneseNameRecognize) {
            JapanesePersonRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
        }
        if (config.placeRecognize) {
            PlaceRecognition.Recognition(vertexList, wordNetOptimum, wordNetAll);
        }
        if (config.organizationRecognize) {
            // 层叠隐马模型——生成输出作为下一级隐马输入
            vertexList = Dijkstra.compute(GenerateBiGraph(wordNetOptimum));
            wordNetOptimum.addAll(vertexList);
            OrganizationRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
        }
        if (!NERexists && preSize != wordNetOptimum.size()) {
            NERexists = true;
        }
    }
}
```

}

}

...

3.2.4 节以中国人名词典为例介绍了 HMM 类型的词典结构。下面简要看一下命名实体的基本算法过程。中国人名识别的算法由 `PersonRecognition` 类的 `Recognition` 方法完成。函数总体上分为三个部分，代码中的黑体字部分为调试输出的查看内容。

(1) 查询句子中每个词的人名识别标签。

```
public static boolean Recognition(List<Vertex> pWordSegResult, WordNet
wordNetOptimum, WordNet wordNetAll)
{
    List<EnumItem<NR>> roleTagList = roleTag(pWordSegResult); //返回当前词对
    应的人名标签列表
    if (HanLP.Config.DEBUG)
    {
        StringBuilder sbLog = new StringBuilder();
        Iterator<Vertex> iterator = pWordSegResult.iterator();
        for (EnumItem<NR> nrEnumItem : roleTagList)
        {
            sbLog.append('[');
            sbLog.append(iterator.next().realWord);
            sbLog.append(' ');
            sbLog.append(nrEnumItem);
            sbLog.append(']');
        }
        System.out.printf("人名角色观察: %s\n", sbLog.toString());
    }
    ...
}
```

将输入的句子查询后缀为 `trie.dat` 的人名词典，得到粗分结果中每个词对应的人名标签和词频。执行输出，代码如下。

人名角色观察:

```
[ A 42634591 ]
[石 B 798 C 177 D 140 E 100 K 2 L 1 ]
[国 C 2368 D 1390 B 51 E 33 L 4 ]
[样 D 1473 C 484 E 110 K 1 L 1 ]
[会见 L 63 K 12 M 3 ]
[乔布斯 A 42634591 ]
[说 L 10922 K 186 M 40 E 9 C 1 ]
[iphone A 42634591 ]
```



```
[是 K 2507 L 2504 M 123 C 10 E 1 ]
[最好 L 4 K 2 ]
[用 L 363 K 73 C 41 D 25 M 2 ]
[的 L 15411 K 11354 M 96 C 1 ]
[手机 L 29 ]
[。 L 3667 ]
[ A 42634591 ]
```

(2) 查询后缀为 value.dat 的人名 HMM 词典, 使用 Viterbi 算法得到最有可能的人名标签。

```
...
List<NR> nrList = viterbiExCompute(roleTagList); // Viterbi 算法

if (HanLP.Config.DEBUG)
{
    StringBuilder sbLog = new StringBuilder();
    Iterator<Vertex> iterator = pWordSegResult.iterator();
    sbLog.append('[');
    for (NR nr : nrList)
    {
        sbLog.append(iterator.next().realWord);
        sbLog.append('/');
        sbLog.append(nr);
        sbLog.append(" ,");
    }
    if (sbLog.length() > 1) sbLog.delete(sbLog.length() - 2, sbLog.length());
    sbLog.append(']');
    System.out.printf("人名角色标注: %s\n", sbLog.toString());
}
...
```

输出结果如下。

```
人名角色标注: [ /A , 石/B , 国/C , 祥/D , 会见/L , 乔布斯/A , 说/K , iphone/A , 是/K , 最好/L ,
用/K , 的/L , 手机/L , 。/L , /A]
```

(3) 使用 parsePattern 解析出最终的人名模式。

```
...
//解析出最终的模式
PersonDictionary.parsePattern(nrList, pWordSegResult, wordNetOptimum,
wordNetAll);
return true;
}
```

`PersonDictionary` 类的 `parsePattern` 是一个很大的方法。本节仅给出简要的介绍，而不探讨内部的细节，原因如下。(1) 本节的内容主要集中于分词算法，对于一些辅助函数，因为实现的多样化，没有必要详细说明。(2) 命名实体类的算法近来变化较大，此类算法的代表性不强，仅对小样本、专业问题的解决可能会有良好的效果，若读者感兴趣，则可以参考 HanLP 的相关文档。

该方法的主要功能是根据识别出的元模式构成最有可能的人名模式，简而言之，就是得到从元模式到人名模式的一个映射函数。

执行结果如下。

识别出人名：石国 BC

识别出人名：石国祥 BCD

这里需要说明的是，解析过程并不简单，还调用了 Aho Corasick 自动机，有兴趣的读者可参考“<http://www.hankcs.com/program/algorithm/aho-corasick-double-array-trie.html>”中的文章。

3.3.10 细分阶段与最短路径

下面讲解分词流程的最后一个环节：细分阶段。

在粗分阶段，把输入的字符串通过一元词典进行原子切分，然后通过二元词典进行消歧，得到了一个基于词典的分词结果。在命名实体识别阶段，进一步对这个结果进行人名、地名、组织机构名的识别，找到了原有句子中的命名实体。

现在，需要将这些结果合并起来，完成最终的分词流程。细分阶段的代码片段如下。

```
...
List<Vertex> vertexList = coarseResult.get(0);
    if (NERexists)
    {
        Graph graph = GenerateBiGraph(wordNetOptimum);
        vertexList = Dijkstra.compute(graph);
        if (HanLP.Config.DEBUG)
        {
            System.out.printf("细分词网%s\n", wordNetOptimum);
            System.out.printf("细分词图%s\n", graph.printByTo());
        }
    }
...
```

```
return convert(vertexList, config.offset);
```

```
}
```

细分词网与未进行命名实体识别阶段的细分词网的不同如下。

细分词网:

0:[始##始]

1:[石 石][石国 未##人][石国祥 未##人]

2:[国 国]

3:[祥 祥]

4:[会见 会见]

5:[见 见]

6:[乔布斯 未##人]

7:

8:

9:[说 说]

10:[iphone 未##串]

11:

12:

13:

14:

15:

16:[是 是]

17:[最好 最好][最 最]

18:[好 好]

19:[用 用]

20:[的 的]

21:[手机 手机]

22:

23:[。 。]

24:[未##末]

这里明显多了[石国 未##人]、[石国祥 未##人]两个新词。这是来自命名实体识别的结果。将新结果加入一元词网后,与粗分阶段相同,执行 `GenerateBiGraph` 方法得到二元词图,代码如下。

细分词图: =====按终点打印=====

to: 1, from: 0, weight:04.60, word:始##始@石

to: 2, from: 0, weight:03.19, word:始##始@未##人

to: 3, from: 0, weight:03.19, word:始##始@未##人

to: 4, from: 1, weight:11.36, word:石@国

to: 5, from: 2, weight:11.61, word:未##人@祥

to: 5, from: 4, weight:10.85, word:国@祥


```

to: 6, from: 3, weight:04.78, word:未##人@会见
to: 6, from: 5, weight:11.57, word:祥@会见
to: 8, from: 6, weight:02.92, word:会见@未##人
to: 8, from: 7, weight:10.91, word:见@未##人
to: 9, from: 8, weight:06.60, word:未##人@说
to: 10, from: 9, weight:04.02, word:说@未##串
to: 11, from: 10, weight:06.80, word:未##串@是
to: 12, from: 11, weight:06.06, word:是@最好
to: 13, from: 11, weight:04.67, word:是@最
to: 14, from: 13, weight:09.49, word:最@好
to: 15, from: 12, weight:03.72, word:最好@用
to: 15, from: 14, weight:05.48, word:好@用
to: 16, from: 15, weight:02.71, word:用@的
to: 17, from: 16, weight:05.32, word:的@手机

```

系统使用 Dijkstra 的最短路径法（函数名称为 `compute`）计算得到最终的最优路径。最后使用 `convert` 将结果返回。

执行结果如下。

```

N-最短分词: [石国祥/nr, 会见/v, 乔布斯/nrf, 说/v, iphone/nx, 是/vshi, 最好/d, 用/p, 的
/udel, 手机/n, 。/w]

```

3.4 结语

本章全面介绍了中文分词算法。

3.1 节介绍了中文分词的相关历史, 以及遇到的问题与困难, 通过对《汉语分词规范》的介绍, 使读者了解中文分词与词性标注的若干规范。

3.2 节重点介绍了 ICTCLAS 中文分词系统总体框架和各类数据结构（词典结构），给出了详细的说明。

3.3 节以介绍算法为主, 引入源算法的部分代码片段, 并给出了说明。读者可以下载项目源码, 之后与本书的词典、代码片段一一对照, 结合本章对源码的讲解、分析, 便可毫无困难地掌握 ICTCLAS 的中文分词算法。有助于读者很容易地建立自己的中文分词系统。

第 4 章

NLP 中的概率图模型

从本章开始引入 NLP 的算法体系。需要说明的是，很多机器学习的算法也常用于 NLP 的任务。例如，用朴素贝叶斯进行文本分类、用 SVM 进行语义角色标注（有关算法的详细内容参见笔者所著的《机器学习算法原理与编程实践》），虽然它们在某些 NLP 任务中都实现了很好的效果，但它们都相互独立，没有形成体系。

随着近些年对智能推理和认知神经学的深入研究，人们对大脑和语言的内在机制了解得越来越多，也越来越能从更高层次上观察和认识思维（包含语言）的现象，由此形成一套完整的算法体系。目前最流行的算法思想包含如下两大流派：基于概率论和图论的概率图模型；基于人工神经网络的深度学习理论。

本章所讲的就是其中之一：概率图模型的主要理论及其算法思想。本章将由浅入深地涉及较多的数学内容，因为篇幅所限，略去基础部分的推导细节，对数学感兴趣的读者可参考相关的书籍。

4.1 概率论回顾

下面先回顾一下有关概率论的几个概念。

4.1.1 多元概率论的几个基本概念

1. 联合概率分布

多维随机变量 X_1, \dots, X_n , 可以用联合概率分布 $P(X_1, \dots, X_n)$ 描述其各个状态的概率, 简称为联合概率分布。它是一个定义在所有变量状态空间的笛卡儿乘积之上的函数。

$$P(X_1, \dots, X_n) : \otimes_{i=1}^n \Omega_{x_i} \rightarrow [0, 1]$$

其中, 所有函数值之和为 1, 即

$$\sum_{X_1, \dots, X_n} P(X_1, \dots, X_n) = 1$$

联合概率分布经常被表示为一张表, 其中包含了 $\prod_{i=1}^n |\Omega_{x_i}|$ 个状态组合及其概率值。如果所有变量都只取两个状态, 则联合分布表共有 2^n 项, 刻画了变量之间的各种关系。

2. 边缘概率分布

记 $X = \{X_1, \dots, X_n\}$, Y 是 X 的真子集, 即 $Y \subset X$, $Z = X \setminus Y$ (\setminus : 表示逆集)。则相对于 $P(X)$, Y 的边缘分布 $P(Y)$ 定义为

$$P(Y) = \sum_Z P(X_1, \dots, X_n)$$

从联合分布 $P(X)$ 到边缘分布 $P(Y)$ 的过程称为边缘化。

例 1 考虑中国香港市场上所有的出租房屋, 从中随机抽取一间, 考查其月租(R)和类型(T)这两个随机变量, 月租可分为 4 等: $\{\text{low(低于 2 000 元), medium(2 000} \sim 6\,000 \text{ 元), upper medium(6 000} \sim 12\,000 \text{ 元), high(高于 12 000 元)}\}$; 类型有 3 种: $\{\text{public(公屋), private(私家屋), others(其他)}\}$ 。联合分布和边缘分布如表 4.1 所示。

表 4.1 联合分布和边缘分布

	public	private	others	边缘 (R)
low	0.17	0.01	0.02	0.20
medium	0.44	0.03	0.01	0.48
upper medium	0.09	0.07	0.01	0.17
high	0.00	0.14	0.01	0.15
边缘 (T)	0.70	0.25	0.05	1.00

3. 条件概率分布

(1) 条件概率: 设 A 、 B 为两个随机事件且 $P(B) > 0$, 事件 A 在给定事件 B 发生时的

条件概率定义为

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

直观上, $P(A|B)$ 是在已知 B 发生时, 对 A 发生的信度; 而 $P(A)$ 则是在不知道 B 是否发生时, 对 A 发生的信度, 由上式可得:

$$P(A \cap B) = P(A|B)P(B)$$

这称为概率的乘法定律, 其含义非常直观: 对 A 和 B 同时发生的信度, 等于 B 发生的信度乘以已知 B 发生时对 A 发生的信度。乘法定律可以写为

$$P(A \cap B) = P(A)P(B|A)$$

(2) 条件概率分布: 设 X 和 Y 是两个随机变量, x 和 y 分别是它们的一个取值, 考虑事件 $X=x$ 在给定 $Y=y$ 时的条件概率为

$$P(X=x | Y=y) = \frac{P(X=x, Y=y)}{P(Y=y)}$$

在上式中, 固定 y , 让 x 在 Ω_x 上变动, 则得到一个在 Ω_x 上的函数。该函数称为在给定 $Y=y$ 时变量 X 的条件概率分布, 记为 $P(X|Y=y)$ 。用 $P(X|Y)$ 记 $\{P(X|Y=y) | y \in \Omega_y\}$, 即在 Y 取不同值时 X 的条件概率分布的集合。 $P(X|Y)$ 称为给定 Y 时变量 X 的条件概率分布。在上式中, 让 x 和 y 在 Ω_x 和 Ω_y 上变动, 则得到一组等式, 把这些等式缩写为

$$P(X | Y) = \frac{P(X, Y)}{P(Y)}$$

上式可视为 $P(X|Y)$ 的直接意义。

一般的, 设 $X=\{X_1, \dots, X_n\}$ 和 $Y=\{Y_1, \dots, Y_m\}$ 为两个变量集合, $P(X, Y)$ 为 $X \cup Y$ 的联合概率分布, $P(Y)$ 为 Y 的边缘概率分布, 则给定 Y 时 X 的条件概率分布定义为

$$P(X | Y) = \frac{P(X, Y)}{P(Y)}$$

根据例 1 提出问题。随机抽取一间私家屋, 其租金为 low 的概率为多大? 此即问给定 $T=\text{private}$ 时 $R=\text{low}$ 的条件概率: $P(R=\text{low} | T=\text{private})$ 。按定义, 有:

$$P(R=\text{low} | T=\text{private}) = \frac{P(R=\text{low}, T=\text{private})}{P(T=\text{private})} = \frac{0.01}{0.25} = 0.04$$

若给定 T 时, 变量 R 的条件分布如表 4.2 所示。

表 4.2 条件分布

	public	private	others
low	0.17 / 0.7	0.01 / 0.25	0.02 / 0.05
medium	0.44 / 0.7	0.03 / 0.25	0.01 / 0.05
upper medium	0.09 / 0.7	0.07 / 0.25	0.01 / 0.05
high	0.00 / 0.7	0.14 / 0.25	0.01 / 0.05

转置表 4.2 得到条件概率分布如表 4.3 所示。

表 4.3 条件概率分布

	low	medium	upper medium	high	SUM
public	0.24	0.63	0.13	0.00	1.00
private	0.04	0.12	0.28	0.56	1.00
others	0.40	0.20	0.20	0.20	1.0

4. 独立性与条件独立性

$P(X|Y=y)$ 是已知 $Y=y$ 时，变量 X 的概率分布，而 $P(X)$ 是未知 Y 的取值时 X 的概率分布。所以，变量 x 与 y 相互独立意味着：对变量 Y 的取值的了解不会改变变量 X 的概率分布。同样，对变量 X 的取值的了解也不会改变变量 Y 的概率分布。

一般的，称随机变量 X_1, X_2, \dots, X_n 相互（边缘）独立，如果 $P(X_1, X_2, \dots, X_n) = P(X_1)P(X_2) \dots P(X_n)$ 。

条件独立性定义：假如 $P(A|B \cap C) = P(A|C)$ 或者 $P(B \cap C) = 0$ ，则事件 A 在给定事件 C 时，在分布 P 上条件独立于事件 B ，记作 $P = (A \perp B | C)$ 。也就是说，如果 P 满足 $(A \perp B | C)$ ，当且仅当 $P(A \cap B | C) = P(A | C) P(B | C)$ 。

4.1.2 贝叶斯与朴素贝叶斯算法

贝叶斯公式最早是由英国神学家贝叶斯提出来的，用来描述两个条件概率直接的关系。在之前的条件概率定义中，我们知道

$$P(A | B) = \frac{P(A, B)}{P(B)}, \quad P(B | A) = \frac{P(A, B)}{P(A)}$$

由上式进一步推导得到：

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

由此,推广到随机变量的范畴,设 X, Y 为两个随机变量,得到贝叶斯公式:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

其中, $P(Y)$ 叫作先验概率, $P(Y|X)$ 叫作后验概率, $P(Y, X)$ 是联合概率。

在机器学习的视角下,我们把 X 理解成“具有某种特征”,把 Y 理解成“类别标签”:

$$P(\text{所属类别} | \text{某种特征}) = \frac{P(\text{某种特征} | \text{所属类别})P(\text{所属类别})}{P(\text{某种特征})}$$

贝叶斯方法把计算“具有某特征的条件下属于某类”的概率转换成需要计算“属于某类的条件下具有某特征”的概率,属于有监督学习。

朴素贝叶斯理论源于随机变量的独立性,之所以称之为朴素是因为其思想基础的简单性:就文本分类而言,从朴素贝叶斯的角度来看,句子中的两两词之间的关系是相互独立的,即一个对象的特征向量中每个维度都是相互独立的。这是朴素贝叶斯理论的思想基础。在多维的情况下,朴素贝叶斯分类的表示为如下内容。

(1) 设 $x = \{a_1, a_2, \dots, a_m\}$ 为一个待分类项,而每个 a 为 x 的一个属性特征。

(2) 有类别集合 $C = \{y_1, y_2, \dots, y_n\}$ 。

(3) 计算 $p(y_1|x), p(y_2|x), \dots, p(y_n|x)$ 。

(4) 如果 $p(y_k|x) = \max \{p(y_1|x), p(y_2|x), \dots, p(y_n|x)\}$, 则 $x \in y_k$ 。

那么,现在的关键就是如何计算第(3)步中的各个条件概率。

(1) 找到一个已知分类的待分类项集合,也就是训练集。

(2) 统计得到在各类别下各个特征属性的条件概率估计,即

$$p(a_1|y_1), p(a_2|y_1), \dots, p(a_m|y_1);$$

$$p(a_1|y_2), p(a_2|y_2), \dots, p(a_m|y_2);$$

.....

$$p(a_m|y_n), p(a_m|y_n), \dots, p(a_m|y_n)。$$

(3) 如果各个特征属性是条件独立的(或者假设它们之间是相互独立的),则根据贝叶斯定理有如下推导:

$$p(y_i|x)=\frac{p(x|y_i)p(y_i)}{p(x)}$$

因为分母对于所有类别为常数,只要将分子最大化皆可。又因为各特征属性是条件独立的,所以有:

$$p(x|y_i)p(y_i)=p(a_1|y_i)p(a_2|y_i)\dots p(a_m|y_i)p(y_i)=p(y_i)\prod_{j=1}^m p(a_j|y_i)$$

根据上述分析,朴素贝叶斯分类的流程可以表示如下。

第一阶段,训练数据生成训练样本集:TF-IDF。

第二阶段,对每个类别计算 $P(y_i)$ 。

第三阶段,对每个特征属性计算所有划分的条件概率。

第四阶段,对每个类别计算 $p(x|y_i)p(y_i)$ 。

第五阶段,以 $p(x|y_i)p(y_i)$ 的最大项作为 x 的所属类别。

4.1.3 文本分类

在处理文本分类之前,引入如下两个概念。

1. One-Hot 表达

文本分类的结构化方法就是 One-Hot 表达 (Representation) 模型。它是最直观,也是目前为止最常用的词表示方法,虽然越来越多的实践已经证明,这种模型存在局限性,但它仍在文本分类中得到广泛应用。

假设把语料库中的所有词都收集为一个词典 D ,词典容纳了语料库中所有句子的词汇。One-Hot 方法就是把每个词表示为一个长长的向量。这个向量的维度是词典大小,其中绝大多数元素为 0,只有一个维度的值为 1。这个维度就代表了当前的词。例如,一个语料库有三个文本,代码如下。

```
文本 1: My dog ate my homework.  
文本 2: My cat ate the sandwich.  
文本 3: A dolphin ate the homework.
```

这三个文本生成的词典共有 9 个词，代码如下。

```
[a, ate, cat, dolphin, dog, homework, my, sandwich, the]
```

这 9 个词依次表示为 One-Hot 向量的形式如下。

"a"	→	[1 0 0 0 0 0 0 0 0]
"ate"	→	[0 1 0 0 0 0 0 0 0]
...
"the"	→	[0 0 0 0 0 0 0 0 1]

其中，每个词都是“茫茫 0 海”中的一个 1。这种稀疏的表达方式就是 One-Hot 表达。它的特点是相互独立地表示语料中的每个词。词与词在句子中的相关性被忽略了，这正符合朴素贝叶斯对文本的假设。

2. 权重策略：TF-IDF 方法

由 One-Hot 向量形式构成的句子就是一个词袋模型，它将文本中的词和模式串转换为数字，而整个文本集也都转换为维度相等的词向量矩阵。

仍旧使用上述三个文本，直观上，文本的词向量可以使用二值表示，内容如下。

文本 1: 0,1,0,0,1,1,1,0,0 (注意，尽管“my”出现了两次，但二元向量表示中仍然是“1”)。

文本 2: 0,1,1,0,0,0,1,1,1。

文本 3: 1,1,0,1,0,1,0,0,1。

这种方式的问题是忽略了一个句子中出现多个相同词的词频信息，增加这些词频信息，就变成了整型计数方式。使用整型计数方式的词向量表示如下。

文本 1: 0,1,0,0,1,1,2,0,0 (请注意，“my”在句子中出现了两次)。

文本 2: 0,1,1,0,0,0,1,1,1。

文本 3: 1,1,0,1,0,1,0,0,1。

接下来，对整型计数方式进行归一化。归一化可以避免句子长度不一致问题，便于算法计算，而且对于基于概率算法，词频信息就变为了概率分布。这就是文档的 TF 信息，内容如下。

文本 1: 0,1/5,0,0,1/5,1/5,2/5,0,0 (请注意，“my”在句子中出现了两次)。

文本 2: 0,1/5,1/5,0,0,0,1/5,1/5,1/5。

文本 3: 1/5,1/5,0,1/5,0,1/5,0,0,1/5。

但是这里还有一个问题，如何体现生成的词袋中的词频信息呢？

原信息: a (1), ate (3), cat (1), dolphin (1), dog (1), homework (2), my (3), sandwich (1), the (2)。

注意：由于词袋收集了所有文档中的词，这些词的词频是针对所有文档的词频，因此，词袋的统计基数是文档数。

词条的文档频率: a (1/3), ate (3/3), cat (1/3), dolphin (1/3), dog (1/3), homework (2/3), my (2/3), sandwich (1/3), the (2/3)。

词袋模型的 IDF 权重如下。

IDF: a $\log(3/1)$, ate $\log(3/3)$, cat $\log(3/1)$, dolphin $\log(3/1)$, dog $\log(3/1)$, homework $\log(3/2)$, my $\log(3/2)$, sandwich $\log(3/1)$, the $\log(3/2)$ 。

TF-IDF 权重策略：计算文本的权重向量，应该选择一个有效的权重方案。最流行的方案是对 TF-IDF 权重的方法。TF-IDF 的含义是词频—逆文档频率，其含义是如果某个词或短语在一篇文章中出现的频率 TF 高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。在前面的例子中，在文本 1 “my” 的词频是 2，因为它在文档中出现了两次。而在这三个文件集合中，“my” 词条的文档频率也是 2。TF-IDF 的假设是，高频率词应该具有高权重，除非它也是高文档频率。“my” 这个词在文本中是经常出现的词汇。它不仅多次发生在单一的文本中，几乎也发生在每个文档中。逆文档频率就是使用词条的文档频率来抵消该词的词频对权重的影响，而得到一个较低的权重。

下面给出定义和公式（来自百度百科）。

词频（Term Frequency, TF）是指某一个给定的词语在该文件中出现的频率。这个数字是对词数（Term Count）的归一化，以防止它偏向长的文件。对于在某一特定文件中的词语来说，它的重要性可表示为：

$$TF_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

上述式子中分子是该词在文件中的出现次数，而分母则是在文件中所有字词的出现次数之和。

逆向文件频率 (Inverse Document Frequency, IDF) 是一个词语普遍重要性的度量。某一特定词语的 IDF, 可以由总文件数目除以包含该词语之文件的数目, 再将得到的商取对数得到:

$$\text{IDF}_i = \log \frac{|D|}{|\{j: t_i \in d_j\}|}$$

其中, $|D|$: 语料库中的文件总数; j : 包含词语的文件数目, 如果该词语不在语料库中, 就会导致分母为零。因此, 一般情况下使用 $1+|\{d \in D: t \in d\}|$ 作为分母; 然后再计算 TF 与 IDF 的乘积。

某一特定文件内的高词语频率, 以及该词语在整个文件集合中的低文件频率, $\text{TFIDF}_{ij} = \text{TF}_{ij} \times \text{IDF}_{ij}$ 可以产生出高权重的 TF-IDF。因此, TF-IDF 倾向于过滤掉常见的词语, 保留重要的词语。

4.1.4 文本分类的实现

首先创建名为 Nbayes_lib.py 的文件, 这个文件用来编写导入的数据和朴素贝叶斯类的代码。

为了将主要精力都集中在算法本身, 使用最简单的语料作为数据集, 代码如下。

```
def loadDataSet():
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                  ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
                  ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him', 'my'],
                  ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                  ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
                  ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0,1,0,1,0,1] #1 is abusive, 0 not
    return postingList,classVec
```

postList 是训练集文本, classVec 是每个文本对应的分类。根据前面所讲的步骤, 逐步实现贝叶斯算法的全过程。

编写一个贝叶斯算法类, 并创建默认的构造方法, 代码如下。

```
class NBayes(object):
    def __init__(self):
        self.vocabulary = [] # 词典
        self.idf=0           # 词典的 idf 权值向量
```

```

self.tf=0          # 训练集的权值矩阵
self.tdm=0         #  $P(x|y_i)$ 
self.Pcates = {}   #  $P(y_i)$  是一个类别词典
self.labels=[]     # 对应每个文本的分类, 是一个外部导入的列表
self.doclength = 0 # 训练集文本数
self.vocablen = 0  # 词典词长
self.testset = 0   # 测试集

def train_set(self, trainset, classVec): # 导入和训练数据集, 生成算法必需的参数和数
                                         # 据结构:
    self.cate_prob(classVec) # 计算每个分类在数据集的概率:  $P(y_i)$ 
    self.doclength = len(trainset)
    tempset = set()
    [tempset.add(word) for doc in trainset for word in doc] # 生成词典
    self.vocabulary = list(tempset)
    self.vocablen = len(self.vocabulary)
    self.calc_wordfreq(trainset) # 计算词频数据集
    self.build_tdm()             # 按分类累计向量空间的每维值:  $P(x|y_i)$ 

def cate_prob(self, classVec):          # 计算在数据集中每个分类的概率:  $P(y_i)$ 
    self.labels = classVec
    labeltemps = set(self.labels) # 获取全部分类
    for labeltemp in labeltemps:
        # 统计列表中重复的分类: self.labels.count(labeltemp)
        self.Pcates[labeltemp] = float(self.labels.count(labeltemp)) /
float(len(self.labels))
    # 生成普通的词频向量

def calc_wordfreq(self, trainset):
    self.idf = np.zeros([1, self.vocablen]) # 1*词典数
    self.tf = np.zeros([self.doclength, self.vocablen]) # 训练集文件数*
                                                         # 词典数

    for indx in xrange(self.doclength): # 遍历所有的文本
        for word in trainset[indx]:     # 遍历文本中的每个词
            self.tf[indx, self.vocabulary.index(word)] += 1

    # 找到文本的词在词典中的位置+1
    for signleword in set(trainset[indx]):
        self.idf[0, self.vocabulary.index(signleword)] += 1

    # 按分类累计向量空间的每维值:  $P(x|y_i)$ 

def build_tdm(self):
    self.tdm = np.zeros([len(self.Pcates), self.vocablen]) # 类别行*词典列
    sumlist = np.zeros([len(self.Pcates), 1]) # 统计每个分类的总值
    for indx in xrange(self.doclength):

```

```

        self.tdm[self.labels[indx]] += self.tf[indx] # 将同一类别的词向
                                                    # 量空间值加总

        # 统计每个分类的总值——是一个标量
        sumlist[self.labels[indx]] =
np.sum(self.tdm[self.labels[indx]])

        self.tdm = self.tdm/sumlist # 生成  $P(x|y_i)$ 

def map2vocab(self, testdata): # 将测试集映射到当前词典
    self.testset = np.zeros([1, self.vocablen])
    for word in testdata:
        self.testset[0, self.vocabulary.index(word)] += 1

def predict(self, testset): # 预测分类结果, 输出预测的分类类别
    if np.shape(testset)[1] != self.vocablen: # 如果测试集长度与词典不相等,
                                                # 则退出程序

        print "输入错误"
        exit(0)

    predvalue = 0 # 初始化类别概率
    predclass = "" # 初始化类别名称
    for tdm_vect, keyclass in zip(self.tdm, self.Pcates):
        #  $P(x|y_i)$   $P(y_i)$ 
        temp = np.sum(testset*tdm_vect*self.Pcates[keyclass])

# 变量 tdm, 计算最大分类值
        if temp > predvalue:
            predvalue = temp
            predclass = keyclass

    return predclass

# 生成 tf-idf
def calc_tfidf(self, trainset):
    self.idf = np.zeros([1, self.vocablen])
    self.tf = np.zeros([self.doclength, self.vocablen])
    for indx in xrange(self.doclength):
        for word in trainset[indx]:
            self.tf[indx, self.vocabulary.index(word)] += 1
        # 消除不同句长导致的偏差
        self.tf[indx] = self.tf[indx]/float(len(trainset[indx]))
        for signleword in set(trainset[indx]):
            self.idf[0, self.vocabulary.index(signleword)] += 1
    self.idf = np.log(float(self.doclength)/self.idf)
    self.tf = np.multiply(self.tf, self.idf) # 矩阵与向量的点乘  $tf \times idf$ 

```

执行代码如下。

```
# -*- coding: utf-8 -*-
```



```
import sys
import os
from numpy import *
import numpy as np
from Nbayes_lib import *

dataSet, listClasses = loadDataSet() # 导入外部数据集
# dataset: 句子的词向量,
# listClass 是句子所属的类别 [0,1,0,1,0,1]
nb = NBayes() # 实例化
nb.train_set(dataSet, listClasses) # 训练数据集
nb.map2vocab(dataSet[0]) # 随机选择一个测试句
print nb.predict(nb.testset) # 输出分类结果
```

分类结果如下。

1

4.2 信息熵

如果说概率是对事件确定性的度量，那么信息（包括信息量和信息熵）就是对事物不确定性的度量。信息熵是由香农（C. E. Shannon）在 1948 年发表的论文《通信的数学理论（A Mathematical Theory of Communication）》中提出的概念。他借用热力学中热熵的概念（热熵是表示分子状态混乱程度的物理量），解决了对信息的量化度量问题，也常用来对不确定性进行度量。

4.2.1 信息量与信息熵

信息量在数学上表示为 $I(X) = -\log P(X)$

信息熵则被定义为对平均不确定性的度量。一个离散随机变量 X 的信息熵 $H(X)$ 定义为：

$$H(X) = \sum_x P(X) \log \frac{1}{P(X)} = -\sum_x P(X) \log P(X)$$

其中，约定 $0\log(1/0)=0$ ，对数若以 2 为底，则熵的单位是比特；若以 e 为底，则其单位是奈特。若无特殊说明，则本书以后章节均采用比特为单位。

- 信息熵的本质是信息量的期望。
- 信息熵是对随机变量不确定性的度量。随机变量 X 的熵越大, 说明它的不确定性也越大。若随机变量退化为定值, 则熵为 0。
- 平均分布是“最不确定”的分布。

下面举两个例子来说明这一点。

例 1 图 4.1 考虑一个取值为 0 或 1 的随机变量 X , 满足(0,1)分布, 记 $p=P(x=1)$ 。根据熵的定义, 有:

$$H(X) = -p \log p - (1-p) \log(1-p)$$

如图 4.1 所示, 当 $p=0$ 或 $p=1$ 时, 我们肯定地知道 X 的取值, 不确定性最小, $H(X)=0$ 。当 $p=0.5$ 时, 对 X 的取值的不确定性达到最大, 此时 $H(X)=1$ 。该例子验证了上文中熵的性质。

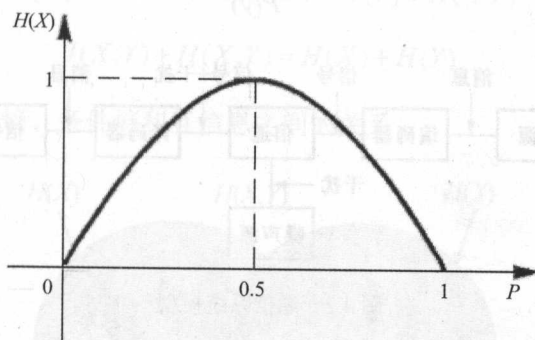


图 4.1 (0,1)分布下随机变量的熵 ($p=P(X=1)$)

例 2 记 X 、 Y 和 Z 分别为掷硬币、掷骰子, 以及从 54 张扑克牌中随意抽取一张的结果。显然 X 的不确定性最小, Y 的不确定性居中, 而 Z 的不确定性最大。与之相应, 这 3 个随机变量的熵之间也恰恰存在这样的关系, 即 $H(X) < H(Y) < H(Z)$ 。

$$H(X) = \sum_{i=1}^6 \log 6 = \log 6$$

$$H(Y) = \sum_{i=1}^6 \log 6 = \log 6$$

$$H(Z) = \sum_{i=1}^{54} \frac{1}{54} \log 54 = \log 54$$

用 $|X|$ 来记作 X 变量的取值个数, 又称为变量的势。

熵的基本性质如下。

(1) $H(X) \geq 0$ 。

(2) $H(X) \leq \log|X|$ ，等号成立的条件，当且仅当 X 的所有取值 x 有 $P(X=x)=1/|X|$ 。

4.2.2 互信息、联合熵、条件熵

1. 互信息

如图 4.2 所示，回到第 2 章出现的这幅图，通过该图理解互信息比较容易。一般而言，信道中总是存在着噪声和干扰，信源发出消息 x ，通过信道后信宿只可能收到由于干扰作用引起的某种变形 y 。信宿收到 y 后推测信源发出 x 的概率，这一过程可由后验概率 $p(x|y)$ 来描述。相应的，信源发出 x 的概率 $p(x)$ 称为先验概率。定义 x 的后验概率与先验概率比值的对数为 y 对 x 的互信息量（简称互信息）。其公式如下。

互信息定义： $I(y,x) = I(y) - I(y|x) = \log \frac{P(y|x)}{P(y)}$

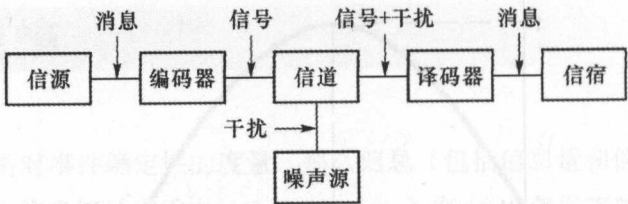


图 4.2 通信系统模型

互信息的性质如下。

- (1) 互信息可以理解为，受信者收到信息 X 后，对信源 Y 的不确定性的消除。
- (2) 互信息 = $I(\text{先验事件}) - I(\text{后验事件}) = \log \frac{\text{后验概率}}{\text{先验概率}}$ 。
- (3) 互信息是对称的。

平均互信息： $I(X;Y) = \sum_{x,y} P(X,Y) \log \frac{P(X,Y)}{P(X)P(Y)}$ 。

平均互信息又称为信息增益。

2. 联合熵

联合熵是借助联合概率分布对熵的自然推广，两个离散随机变量 X 和 Y 的联合熵定义为

$$H(X, Y) = \sum_{X, Y} P(X, Y) \log \frac{1}{P(X, Y)} = - \sum_{X, Y} P(X, Y) \log P(X, Y)$$

3. 条件熵

条件熵是利用条件概率分布对熵的一个延伸。随机变量 X 的熵是用它的概率分布 $P(X)$ 来定义的。如果知道另一个随机变量 Y 的取值为 y , 那么 X 的后验分布即为 $P(X|Y=y)$ 。利用条件分布可以定义给定 $Y=y$ 时 X 的条件熵为

$$H(X|Y=y) = - \sum_X P(X|Y=y) \log P(X|Y=y)$$

熵 $H(X)$ 度量的是随机变量 X 的不确定性, 条件熵 $H(X|Y=y)$ 度量的则是已知 $Y=y$ 后, X 的不确定性。

熵的链式规则:

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

$$I(X; Y) + H(X, Y) = H(X) + H(Y)$$

图 4.3 所示为联合熵、条件熵和互信息之间的关系。

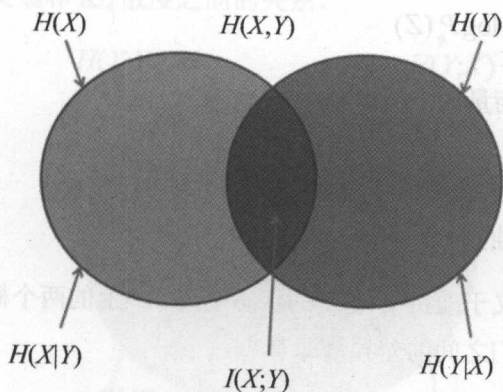


图 4.3 联合熵、条件熵和互信息之间的关系

边缘独立定理: 从信息论角度为“边缘独立”这一概念提供了一个直观解释, 即两个随机变量相互独立当且仅当它们之间的互信息为 0。

接下来考虑 3 个变量 X 、 Y 和 Z 之间的条件独立关系。条件熵 $H(X|Z)$ 是给定 Z 时 X 剩余的不确定性, 如果进一步再给定 Y , 则 X 剩余的不确定性变为 $H(X|Z, Y)$ 。因此, 这两者之差即是给定 Z 时观测 Y 的取值会带来的关于 X 的信息量, 即

$$I(X; Y|Z) = H(X|Z) - H(X|Z, Y)$$

称为给定 Z 时 Y 关于 X 的信息, 容易证明 $I(X;Y|Z) = I(Y;X|Z)$ 。于是, $I(X;Y|Z)$ 也称为给定 Z 时 X 和 Y 之间的条件互信息。

定理 4-1: 对任意 3 个离散随机变量 X 、 Y 和 Z , 有

$$(1) I(X,Y|Z) \geq 0.$$

$$(2) H(X|Y,Z) \leq H(X|Z).$$

上述定理的意义在于, 它从信息论角度为随机变量之间的“条件独立”这一概念提供了一个直观解释, 即给定 Z , 两个随机变量 X 和 Y 相互条件独立, 当且仅当它们的条件互信息为零, 或者说, Y 关于 X 的信息已全部包含在 Z 中, 从而观测到 Z 后, 再对 y 进行的观测不会带来关于 X 的更多信息。另外, 如果 X 和 Y 在给定 Z 时相互不独立, 则 $H(X|Z,Y) < H(X|Z)$, 即在已知 Z 的基础上对 Y 的进一步观测将会带来关于 X 的新信息, 从而降低 X 的不确定性。

4.2.3 交叉熵和 KL 散度

1. 交叉熵

$$H(P;Q) = -\sum P_p(Z) \log P_q(Z)$$

(1) 交叉熵常用来衡量两个概率分布的差异性。

(2) 在 Logistic 中的交叉熵为其代价函数。

2. 相对熵与变量独立

相对熵定义: 对定义于随机变量 X 的状态空间 Ω_x 上的两个概率分布 $P(X)$ 和 $Q(X)$, 可以用相对熵来度量它们之间的差异, 即有

$$KL(P,Q) = \sum_x P(X) \log \frac{P(X)}{Q(X)}$$

其中, 约定 $0 \log \frac{0}{q} = 0$; $p \log \frac{p}{0} = \infty$, $\forall p > 0$ 。 $KL(P,Q)$ 又称为 $P(X)$ 和 $Q(X)$ 之间的 KL 散度。但严格来讲它不是一个真正意义上的距离, 因为 $KL(P,Q) \neq KL(Q,P)$ 。

定理 4-2: 设 $P(X)$ 和 $Q(X)$ 为定义在某个变量 X 的状态空间 Ω_x 上的两个概率分布, 则有:

$$KL(P,Q) \geq 0$$

其中，当且仅当 P 与 Q 相同，即 $P(X=x)=Q(X=x), \forall x \in \Omega_x$ 时等号成立。

推论，对于满足 $\sum f(X)>0$ 的非负函数 $f(X)$ ，定义概率分布 $P^*(X)$ 为：

$$P^*(X)=\frac{f(X)}{\sum_X f(X)}$$

那么，对于任意其他的概率分布 $P(X)$ ，则有：

$$\sum_X f(X)\log P^*(X)\geqslant \sum_X f(X)\log P(X)$$

其中，当且仅当 P^* 与 P 相同时等号成立。

定理 4-3：互信息与变量独立之间的两个关系。首先有如下定理，对任意两个离散随机变量 X 和 Y 有：

- (1) $I(X,Y)\geqslant 0$ 。
- (2) $H(X|Y)\leqslant H(X)$ 。

上述两式当且仅当 X 与 Y 相互独立时等号成立。

图 4.4 所示为交叉熵和 KL 散度之间的关系。

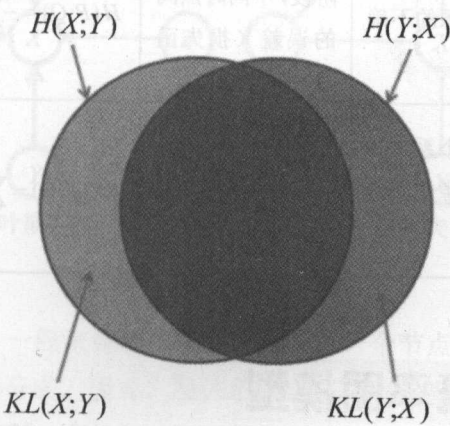


图 4.4 交叉熵和 KL 散度之间的关系

4.2.4 信息熵的 NLP 的意义

对于任何语言系统的抽象模型都是一个信息系统，引入信息熵的本质意义在于从信息论的角度来考察一个语言系统，并且对其行为（编码和解码）提供了统一的测度。这个测度的各个方面都可以通过表 4.4 来说明。

表 4.4 信息熵概念与公式表

名 称	节 点	信息论含义	NLP含义	公 式
X	信源	信源	特征	—
Y	信宿	信宿	标签	—
信息量	信源	离散信源信号发生的不确定性	特征的不确定性	$I(X) = -\log P(X)$
信息熵	信源	离散信源信号发生的平均不确定性	特征不确定性的均值	$H(X) = -\sum_x P(X) \log P(X)$
联合熵	信源->信宿	信息系统整体的不确定性	训练集总体不确定性; 相当于联合概率	$H(X, Y) = -\sum_{X,Y} P(X, Y) \log P(X, Y)$
互信息	信源->信宿	信源到信宿输出的不确定性	特征对应标签的不确定性; 衡量了对称性	$I(X; Y) = \sum_{X,Y} P(X, Y) \log \frac{P(X, Y)}{P(X)P(Y)}$
条件熵	信源->信宿	给定输出结果情况下, 不同的信源的; 非对称	给定标签的情况下, 特征的不确定性, 衡量了差异性	$H(X Y=y) = -\sum_x P(X Y=y) \log P(X Y=y)$
交叉熵	信息系统	衡量两个系统对同一信源的不确定性	某个系统, 在学习阶段, 不同时间点间的误差 (损失函数); 非对称	$H(P; Q) = -\sum P_p(Z) \log P_q(Z)$
KL散度	信息系统	衡量两个系统对同一信源的不确定性	某个系统, 在学习阶段, 不同时间点间的误差 (损失函数); 非对称	$KL(P, Q) = \sum_x P(X) \log \frac{P(X)}{Q(X)}$

4.3 NLP 与概率图模型

概率图模型结合了概率论与图论的知识，用图模式（节点和边）表达基于概率相关关系的模型的总称。最早由图灵奖获得者 Pearl 开发出来。其动机来源于建立一套领域无关的通用自动（智能）推理理论，从中揭示智能推理的内在机制。

我们解决非确定性问题的传统思路都是利用概率论的思想，但是随着问题的复杂性不断增加，传统的概率方法显得越来越力不从心。图模型的引入使人们可以将复杂问题

得到适当的分解：其中，变量表示为节点，变量与变量之间的关系表示为边（或弧），这样就使问题得以结构化。然后，根据图的结构进行训练和计算推理得出最终的结果。因此，概率图理论就自然地分为三个部分，分别为：概率图模型表示理论、概率图模型推理理论和概率图模型学习理论。

4.3.1 概率图模型的几个基本问题

前文已经提到，常用的概率图模型，无论是最简单的朴素贝叶斯模型还是比较复杂的最大熵、条件随机场模型，都包含如下三个基本的问题。

□ 模型的表示。

表示理论将图模型分为如下两个类别：贝叶斯网络（Bayesian Network）和马尔科夫随机场（Markov Random Field）。其中，贝叶斯网络采用有向无环图（Directed Acyclic Graph）来表达事件的因果关系，而马尔科夫随机场则采用无向图（Undirected Graph）来表达变量间的相互作用。

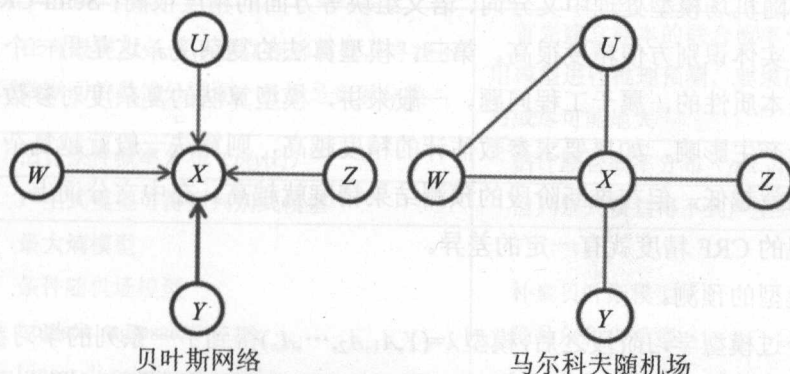


图 4.5 有向图与无向图

如图 4.5 所示，一般来说，贝叶斯网络中每一个节点都对应于一个先验概率分布或者条件概率分布。因此，整体的联合分布可以直接分解为所有单个节点所对应的分布的乘积。有关贝叶斯网络的细节知识建议参考笔者的《机器学习算法原理与编程实践》一书，本节不详细说明。对于马尔科夫随机场，由于变量之间没有明确的因果关系，它的联合概率分布通常会表达为一系列势函数（Potential Function）的乘积。通常情况下，这些乘积的积分并不等于 1。因此，还要对其进行归一化才能形成一个有效的概率分布。在 NLP 中，最常用到的就是各种基于马尔科夫性的各种概率图模型。本章介绍的隐马尔科夫模型、最大熵模型和条件随机场都是从马尔科夫模型发展出来的。

□ 模型的学习。

所谓的学习是指将给定的图模型，首先形式化为数学公式：模型 $\lambda=(Y, A_1, A_2, \dots, A_n)$ ，其中， $A_i \in A_n$ 是模型的参数，标记符号 Y 也称为状态或状态序列，即使用的标签：词性列表、组块标签列表、BIOES 标注等。在自然语言处理中，必须确定使用的语料资源库，即学习所需的样本资源库（也叫作训练集），数学上形式化为 $O=\{o_1, o_2, \dots, o_T\}$ ，其中， $o_i \in O_T$ 是模型的输入序列，也称为观测序列。根据训练集可以得到每个观测序列对应不同标签(Y)的概率分布： $P(O|\lambda)$ 。然后套入模型，估计出模型的 $\lambda=(Y, A_1, A_2, \dots, A_n)$ 。其中， $A_i \in A_n$ ，使得模型 λ 得到最大的概率分布： $P^*(O|\lambda)$ 。

因此，模型的学习精度受如下三方面的影响：第一，语料库样本集对总体的代表性。这属于影响算法精度的外因，这部分本章暂不涉及，在后面章节会有详细讲解。第二，模型算法的理论基础及所针对的问题。这是内因之一。不同模型因为原理不同，能够处理的语言问题也不同，比如朴素贝叶斯模型在处理短文本分类方面精度很高；最大熵模型在处理中文词性标注问题上表现很好；条件随机场模型处理中文分词、语义组块等方面的精度很高；Semi-CRF 在处理命名实体识别方便精度很高。第三，模型算法的复杂度。这是另一个内因，但不是本质性的，属于工程问题。一般来讲，模型算法的复杂度对参数估计的精度会产生影响，如果要求参数估计的精度越高，则算法一般就越复杂，学习效率也就越低，但在推断阶段的预测结果精度就越高。在中文分词上，几种不同实现的 CRF 精度就有一定的差异。

□ 模型的预测。

经过模型学习阶段之后，模型 $\lambda=(Y, A_1, A_2, \dots, A_n)$ 得到了一系列的学习参数 $A_i \in A_n$ ，即后验概率为 $p^*(O|\lambda)$ 。此时，模型即可投于实际的应用，也称为预测（推理）阶段，信息论中称为解码问题。预测过程是对一个新的输入样本 ox 通过学习后的模型，选取最大后验概率指向的最有可能的标签或标签序列作为预测结果。形式上表示为根据样本 ox 判定每种观测序列 Y 的概率，那么观测序列的条件概率 $p(Y|ox)$ 中最大的那个 $p^*(Y|ox)$ 就是预测的结果。

对于概率图模型而言，上述三个环节中学习过程是最重要的过程。后面关于模型的章节中，都会重点剖析此阶段的模型构成与实现的细节。

4.3.2 产生式模型和判别式模型

如图 4.6 所示，主要的概率图模型：朴素贝叶斯模型(NB)、隐马尔科夫模型(HMM)、

最大熵模型（ME）和条件随机场（CRF）。描绘了从单类预测到序列数据预测方面的联合概率分布与条件概率分布。

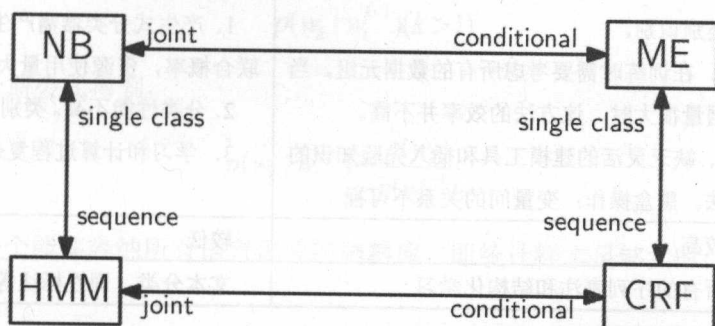


图 4.6 主要的概率图模型中的关系

判别式模型和产生式模型的区别与联系如表 4.5 所示。

表 4.5 判别式模型和产生式模型的区别与联系

	判别式模型（Discriminative Models）	产生式模型（Generative Models）
特 点	在有限样本条件下建立判别函数，寻找不同数据间的最优分类面，目标是实现分类	首先建立样本的联合概率分布，再利用模型进行推理预测。要求已知样本无穷或尽可能地大
区 别	估计条件概率分布 $(p(y x))$	估计联合概率分布 $(p(x,y))$
联 系	产生式模型可得到判别式模型	但判别式模型得不到产生式模型
常见模型	最大熵模型 条件随机场模型 Logistic regression Linear discriminant analysis Support vector machines Boosting Linear regression Neural networks	朴素贝叶斯模型 隐马尔科夫模型 Gaussian mixture model AODE Latent Dirichlet allocation Restricted Boltzmann Machine
优 势	1. 面向分类边界的训练，比使用纯概率方法高级。 2. 能清晰地分辨出类别之间的差异特征。 3. 可用于多类的学习和识别。 4. 判别式模型比产生式模型简单，容易学习	1. 面向整体数据的分布。 2. 能够反映同类数据本身的相似度。 3. 模型可以通过增量学习得到。 4. 可用于数据不完整的情况

续表

	判别式模型 (Discriminative Models)	产生式模型 (Generative Models)
劣势	1. 不能反映训练数据本身的特性, 只能用于类别识别。 2. 在训练时需要考虑所有的数据元组, 当数据量很大时, 该方法的效率并不高。 3. 缺乏灵活的建模工具和插入先验知识的方法。黑盒操作: 变量间的关系不可视	1. 产生式分类器需产生的所有变量的联合概率, 资源使用量大。 2. 分类性能不高。类别识别精度有限。 3. 学习和计算过程复杂
性能	较高	较低
NLP应用	所有的序列标注和结构化学习	文本分类、词性标注等

如果隐马尔科夫模型是朴素贝叶斯模型的序列化模型扩展, 则条件随机场可以理解
为最大熵模型的序列化扩展。这两个最大熵模型和条件随机场模型称为判别式方法。这
些图模型的比较在图 4.6 中已给出。后面的内容将逐一详细介绍隐马尔科夫模型、最大
熵模型和条件随机场模型。

4.3.3 统计语言模型与 NLP 算法设计

在多年自然语言处理算法开发与建模的实践中, 人们逐渐形成一套基于统计理论的
NLP 算法设计方法论。这套方法论最基础的环节就是统计语言模型 (Statistical Language
Model), 它被广泛用于 NLP 的各项任务中。

那么, 什么是语言模型呢? 第 2 章介绍过分词的语言模型, 本章将扩展这个概念到
任意结构的语言模型。简单地说, 统计语言模型是用来计算句子中某种语言模式出现概
率的统计模型。一般自然语言的统计单位是句子, 所以也看作句子的概率模型。假设
 $W=(w_1, w_2, \cdots, w_n)$ 为一个句子, 这个句子有 n 个词, 也就是 n 个词汇按顺序构成的字符序
列, 这里表示为 W_1^n 。前面讲过, 利用贝叶斯公式进行链式分解, w_1, w_2, \cdots, w_n 的联合概
率为

$$\begin{aligned} p(W) &= p(w_1^n) = p(w_1, w_2, \cdots, w_n) \\ &= p(w_1) p(w_2 | w_1) p(w_3 | w_1^2) \cdots p(w_n | w_1^{n-1}) \end{aligned}$$

因为我们扩展了第 2 章的模型, 需要设法计算 $p(w_1) p(w_2 | w_1) p(w_3 | w_1^2) \cdots$
 $p(w_n | w_1^{n-1})$ 这些语言模型的参数, 得到这些参数, 即可得到 $p(w_1^n)$ 。

为了表示方便, 将上述式子表示为语言模型的形式。考虑一个长度为 n 的句子, 位

置为 k 的词出现的概率与其前面的所有的词都相关,也就是说与它前面 $k-1$ 个词都相关,其 k 元语言模型可以表示为

$$p(w_k | w_1^{k-1}) (k > 1)$$

利用贝叶斯公式得到:

$$p(w_k | w_1^{k-1}) = \frac{p(w_k)}{p(w_1^{k-1})} \quad (4.1)$$

假设有一个能够容纳所有语言现象的语料库,即统计样本足够大时, $p(w_k | w_1^{k-1})$ 就变为表达形式:

$$p(w_k | w_1^{k-1}) = \frac{\text{Count}(w_1^k)}{\text{Count}(w_1^{k-1})} \quad (4.2)$$

其中, $\text{Count}(w_1^k)$ 和 $\text{Count}(w_1^{k-1})$ 分别表示 w_k 和 w_1^{k-1} 在语料中出现的次数。

下面来计算一下这个概率。考虑一个给定的长度为 n 的句子就需要计算 n 个参数,不妨假设语料库对应词典 D 的大小(词汇量)为 T 。那么,如果考虑长度为 n 的任意句子上就有 T^n 种可能。而每种可能都要计算 n 个参数,那么总共就需要计算 nT^n 个参数。而这些概率计算好后,还得保存下来。因此,存储这些信息也需要很大的内存开销。当然,这是纯粹数学上的估算,实际情况并不是这样,这涉及语言中的模式问题。假设我们认为某几个汉字成词的可能性仅与它前面的一个词有关,而与它前面两个词和所有后面的词无关(这就是我们后面讲的马尔科夫性),那么当前词与其前面的词就构成了一种二元的语言模型(2-gram)。

这个概念扩展到 n 元,即可得出如下定义:如果一种语言现象出现的概率与它 $n-1$ 个词相关,而与其前面 n 个词和后面的词都无关,那么这就是一个 n -gram 语言模型。将这个模型形式化后,式(4.1)就变为

$$p(w_k | w_1^{k-1}) \approx p(w_k | w_{k-n+1}^{k-1}) \quad (4.3)$$

则式(4.2)变为

$$p(w_k | w_1^{k-1}) \approx \frac{\text{Count}(w_{k-n+1}^k)}{\text{Count}(w_{k-n+1}^{k-1})} \quad (4.4)$$

以 $n=2$ 为例,就有:

$$p(w_k | w_1^{k-1}) \approx \frac{\text{Count}(w_{k-1}, w_k)}{\text{Count}(w_{k-1})} \quad (4.5)$$

n -gram 语言模型中 n 的大小的选取，需要同时考虑计算复杂度和模型效果的两个因素。考虑汉语中词典大小 $D=300\,000$ 个词汇。模型参数数量与 n 的关系如表 4.6 所示。

表 4.6 模型参数数量与 n 的关系

n	模型参数数量
1(unigram)	3×10^5
2(bigram)	9×10^{10}
3(trigram)	27×10^{15}
4(4-gram)	81×10^{20}

模型参数的量级是模型长度 n 的指数函数 $\{O(N^n)\}$ 。显然 n 不能取得太大，实际应用中最多的是采用 $n=3$ 的三元模型。

需要特别说明的是语言模型的平滑问题，内容如下。

- ❑ 若 $\text{Count}(w_{k-n+1}^k)=0$ ，是否认为 $p(w_k|w_1^{k-1})$ 就等于 0 呢？
- ❑ 若 $\text{Count}(w_{k-n+1}^k)=\text{Count}(w_{k-n+1}^{k-1})$ ，是否认为 $p(w_k|w_1^{k-1})$ 就等于 1 呢？

显然不能！但这是一个无法回避的问题，无论你的语料库有多么大，平滑化技术都不可避免，还好，在后面介绍的算法中，这个问题已经得到解决。

总结起来， n -gram 模型的主要工作是在模型的训练阶段，统计语料中各种语言模式（词汇、词性串）出现的次数，并通过一定的统计算法，将它们计算好之后就存储起来。在预测阶段，当需要计算一个新的句子的概率时，只需查找到句子中可能出现的语言模型的概率参数，并将它们连乘起来即可得到最终的结果。

这是 NLP 算法的通用策略，这里涉及两个问题：（1）如何设计语言模型。（2）如何求解算法策略。几乎所有的 NLP 实践都是围绕着这两个问题展开的。

在机器学习领域最常用的策略为：对所考虑的问题建模后，先为其构造一个目标函数，然后对这个目标函数进行优化。从而求得一组最优的参数，最后利用这组最优参数——语言模型来做出预测。

在实践中有些问题受到语言理论的影响，新生成的语言理论改变了人们对原有问题的认识，这样就需要根据新的理论设计新的语言模型。例如，在汉语句法解析的问题上，并行着两种不同的句法理论，包括转换生成语法、依存句法等。对于不同的句法理论需要设计不同的语言模型，再根据语言模型找到不同的算法。

另外,有时候某些算法策略具有坚实的理论基础及较强的普适性。人们希望将这种算法策略能够用于某些新的领域。例如,将深度学习算法用于各种序列标注问题,并与其他算法进行比较优劣,最终做出选择。这种情况下,语言模型(特征)常常在多个算法之间不会改变,需要调整的是使算法适用于特定的语言模型。

4.3.4 极大似然估计

如上所述,在求解机器学习、深度学习、概率图模型中,计算过程一般分为:模型的学习阶段(也称为训练阶段)和模型的预测阶段。

在训练阶段,很多模型的学习函数都常被看作一个最优化问题。模型训练以训练集样本作为已知的总体分布,来计算模型参数过程。该阶段的计算特点是把训练集样本点的取值作为已知量,然后学习出样本点对应的参数,如神经网络中的神经元的权重向量、概率图模型的节点权重值等。

这样,模型的训练过程就变成了统计学中的参数估计过程。在统计学中,参数估计的方法有很多。如果已知某个随机样本的总体概率分布(训练集),但是其中具体的参数不清楚,则参数估计就是通过若干次试验,观察其结果,利用结果推断出参数大概值的方法,称为极大似然估计。

极大似然估计是建立在如下思想上的:已知某个参数能使这个样本出现的概率最大,我们当然不会再去选择其他小概率的样本,所以干脆就把这个参数作为估计的真实值。在讲解后面的内容之前,有必要再回顾一下极大似然估计的基本原理。

极大似然估计:设总体分布为 $f(X, \theta)$, X_1, X_2, \dots, X_n 为该总体采样得到的样本(语料库中的样本集), θ 为待估的参数。因为 X_1, X_2, \dots, X_n 独立同分布,那么它们的联合概率分布为

$$L(x_1, x_2, \dots, x_n; \theta_1, \theta_2, \dots, \theta_k) = \prod_{i=1}^n f(x_i; \theta_1, \theta_2, \dots, \theta_k)$$

因为样本已经存在,这里 x_1, \dots, x_n 都被看作已知量,是固定的, θ 被看作未知的、待估的参数; $L(x, \theta)$ 就变成了 θ 的函数,也就是似然函数。求这个参数 θ 的值,使得似然函数取极大值,这种方法就是极大似然估计。

求极大似然函数估计值的一般步骤如下。

(1) 写出似然函数。

(2) 对似然函数取对数, 并整理。

(3) 求导数。

(4) 解似然方程。

似然函数常写成若干式子连乘积的形式。在实践中, 由于求导数的需要, 最好先将似然函数取对数, 因为函数的对数其极值点不发生变化, 这样就把乘号变为了加号。得到的式子称为对数似然函数。若对数似然函数可导, 可通过求导的方式解出下列方程组, 得到驻点, 然后分析该驻点的极值性。

对于统计语言模型而言, 利用最大似然可把目标函数设为:

$$\prod_{w \in C} p(w | \text{Context}(w))$$

其中, C 表示语料 (Corpus), $\text{Context}(w)$ 表示词 w 的上下文 (Context 或 Window), 即 w 周边的词的集合, 当 $\text{Context}(w)$ 为空时, 就取 $p(w | \text{Context}(w)) = p(w)$ 。特别的, 对于前面介绍的 n -gram 模型有:

$$\text{Context}(w_i) = w_{i-n+1}^{i-1}$$

按照最大对数似然的惯例, 把目标函数设为:

$$L = \sum_{w \in C} \log p(w | \text{Context}(w))$$

然后对这个函数求最大化。由上式可知, 概率 $p(w | \text{Context}(w))$ 已被视为关于 w 和 $\text{Context}(w)$ 的函数, 即

$$p(w | \text{Context}(w)) = F(w, \text{Context}(w), \theta)$$

其中, θ 为待定参数集。这样一来, 一旦对上式进行优化得到最优参数集 θ^* 后, F 也就唯一被确定了, 以后任何概率 $p(w | \text{Context}(w))$ 都可以通过函数 $F(w, \text{Context}(w), \theta^*)$ 来计算。

与 n -gram 相比, 这种方法不需要 (事先计算并) 保存所有的概率值, 而是通过直接计算来获取, 且通过选取合适的模型可使得 θ 中参数的个数远小于 n -gram 中模型参数的个数。很显然, 对于这种方法: 最关键的地方就在于函数 F 的构造。从 4.4 节开始, 将介绍概率图模型中各种构造 F 的方法。

4.4 隐马尔科夫模型简介

有关随机过程、马尔科夫链、隐马尔科夫模型在笔者的《机器学习算法原理与编程实践》一书中已经详细介绍过。为了本书的完整性，这部分仅给出一些定理和定义的简要介绍，并沿用《机器学习算法原理与编程实践》中的部分实例。

4.4.1 马尔科夫链

马尔科夫过程 (Markov Process) 是俄国数学家 A.A. 马尔科夫提出的，其原始模型为马尔科夫链。马尔科夫链是指时间和状态都是离散的马尔科夫过程。该过程具有如下特性：在已知系统当前状态的条件下，它未来的演变不依赖于过去的演变。也就是说，一个马尔科夫过程可以表示为系统在状态转移过程中，第 $T+1$ 次结果只受第 T 次结果的影响，即只与当前状态有关，而与过去状态，即与系统的初始状态和此次转移前的所有状态无关。

其形式化的表示如下。

设有随机过程 $\{X_n, n \in T\}$ ，若对于任意的整数 $n \in T$ 和任意的 $i_0, i_1, \dots, i_{n+1} \in I$ ，条件概率满足：

$$P\{X_{n+1}=i_{n+1}|X_0=i_0, X_1=i_1, \dots, X_n=i_n\}=P\{X_{n+1}=i_{n+1}|X_n=i_n\}$$

则称为马尔科夫链，简称马氏链。

马尔科夫链的一步转移概率，可以定义为

$$\text{条件概率: } P_{ij}(n)=P\{X_{n+1}=j|X_n=i\}$$

为马尔科夫链 $\{X_n, n \in T\}$ 在时刻 n 的一步转移概率，其中 $i, j \in I$ ，简称为转移概率。

一步转移矩阵定义如下。

设 P 表示为一步转移概率 P_{ij} 所组成的矩阵，且状态空间 $I=\{1, 2, 3, \dots\}$ ，则

$$P = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1n} & \dots \\ P_{21} & P_{22} & \dots & P_{2n} & \dots \\ P_{31} & P_{32} & \dots & P_{3n} & \dots \end{pmatrix}$$

马尔科夫链的性质如下。

定理：设 $\{X_n, n \in T\}$ 为一条马尔科夫链，则对任意整数 $n \geq 0$ 和 $0 \leq l \leq n$ $i, j \in I$ ， n 步转

移概率 $p_{ij}^{(n)}$ 具有如下性质。

- (1) $p_{ij}^{(n)} = \sum_{k \in I} p_{ik}^{(1)} p_{kj}^{(n-1)}$ 称为切普曼—柯尔莫戈洛夫方程，简称 C-K 方程，它是马尔科夫链在时域空间中转移概率计算的通用公式。
- (2) $p_{ij}^{(n)} = \sum_{k_1 \in I} \cdots \sum_{k_{n-1} \in I} p_{ik_1} p_{k_1 k_2} \cdots p_{k_{n-1} j}; p_{ik}^{(1)}$ 说明 n 步转移概率完全由一步转移概率决定。
- (3) $P^{(n)} = PP^{(n-1)}$ 。
- (4) $P^{(n)} = P^n$ 说明齐次马尔科夫链的 n 步转移概率矩阵等于一步转移概率矩阵的 n 次乘方。

4.4.2 隐马尔科夫模型

隐马尔科夫模型形成于 20 世纪 60~70 年代，最初在 L. E. Baum 和其他一些学者发表的统计学论文中出现。自 20 世纪 80 年代以来，HMM 应用于语音识别，取得了重大成功。到了 90 年代，HMM 被引入自然语言处理和移动通信核心技术“多用户的检测”，以及生物信息科学、故障诊断等领域体现了很大的价值。

图 4.7 所示为隐马尔科夫模型。一个隐马尔科夫模型（HMM）由两组状态集合和三组概率集合构成。

- ❑ 隐状态：系统的（真实）状态序列，可以由一个马尔科夫过程进行描述。
- ❑ 状态转移矩阵：是隐状态空间，即马尔科夫状态空间的状态转移矩阵。
- ❑ 观察状态：在这个过程中显式状态序列。
- ❑ π 向量：从一个观察状态到每个隐状态的概率向量（初始概率），表示为初始时刻观察状态转换到隐状态的概率。
- ❑ 混淆矩阵：也称为发射概率，包含了每个给定的观察状态转换到每个隐藏状态的概率矩阵。

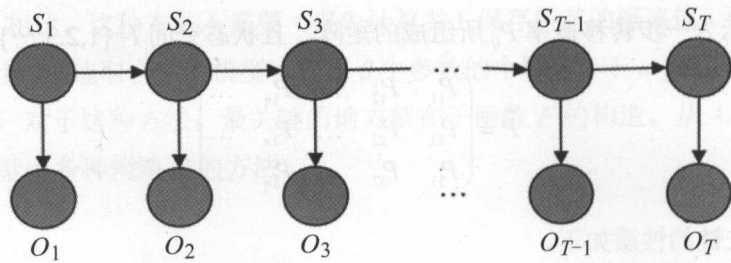


图 4.7 隐马尔科夫模型

根据上述描述, 可以给出隐马尔科夫模型如下的形式化定义。

一个隐马尔科夫模型是一个五元组 (S, O, Π, A, B) 。

(1) S : 一个系统在 t 时刻的状态 (隐状态) 空间集合 (S_1, S_2, \dots, S_n)

(2) O : 一个输出 (观测状态) 状态的集合 (O_1, O_2, \dots, O_m)

(3) $\Pi = (\pi_i)$: 初始化概率向量。

(4) $A = (a_{ij})$: 状态转移矩阵。

(5) $B = (b_{ij})$: 混淆矩阵。

注意, 这里 t 可以不表示时间, 在状态转移矩阵及混淆矩阵中的每个概率都可以与时间无关, 也就是说, 当系统演化时这些矩阵并不一定随时间改变。

对 HMM 来说, 有如下三个重要假设。

假设 1: 马尔科夫假设 (状态构成一阶马尔科夫链)。

$$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | X_{i-1})$$

假设 2: 不动性假设 (状态与具体时间无关)。

$$P(X_{i+1} | X_i) = P(X_{j+1} | X_j), \text{ 对于任意 } i, j$$

假设 3: 输出独立性假设 (输出仅与当前状态有关)。

$$P(O_1, \dots, O_T | X_1, \dots, X_T) = \prod P(O_i | X_i)$$

这三个假设使隐马尔科夫模型摆脱了时域上的限制, 成为一类纯粹的逻辑模型。

4.4.3 HMMs 的一个实例

接下来, 举例求解这个模型。因为比较容易懂, 很多文献中都用到有关天气的例子。例如, 假设预测天气的人住在山洞中, 或者潜水艇里, 反正一个相对封闭的环境中, 无法直接观测天气的变化, 但是他身边有一个湿度计 (这个假设有点离谱), 可以读出每天的干湿度。由于多年的观察, 他还有关于干湿度的统计数据, 他知道一年中晴天、阴天和雨天的总比例, 也就是初始化向量, 还知道每种天气间的转移概率, 以及每类干湿度对应天气的发射概率。具体模型如图 4.8 所示。

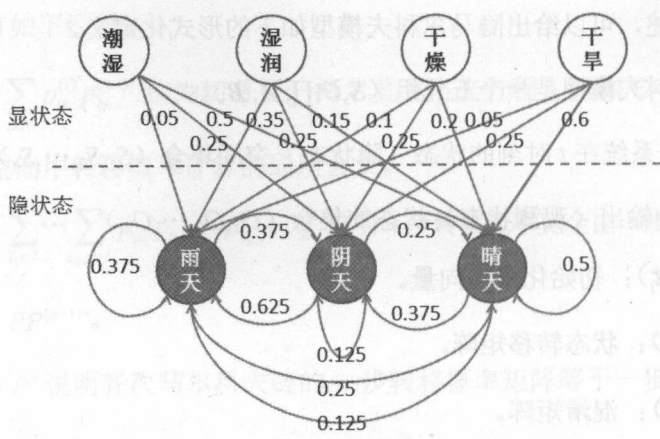


图 4.8 带有概率的天气预测模型

由图 4.8 可知：

- (1) S 是天气系统在 t 时刻的状态空间集合：{晴天，阴天，雨天}。
- (2) O 是一个输出状态的集合：{潮湿，湿润，干燥，干旱}。
- (3) 初始化概率向量 Π ：(0.63, 0.17, 0.20)，表示一年中晴天、阴天、雨天的概率。
- (4) 状态转移矩阵 A （见表 4.7）。

表 4.7 状态转移矩阵

$T \backslash T+1$	晴 天	阴 天	雨 天
晴天	0.5	0.375	0.125
阴天	0.25	0.125	0.625
雨天	0.25	0.375	0.375

- (5) 混淆矩阵 B （发射概率），如表 4.8 所示。

表 4.8 混淆矩阵

$T \backslash T+1$	干 旱	干 燥	湿 润	潮 湿
晴天	0.60	0.20	0.15	0.05
阴天	0.25	0.25	0.25	0.25
雨天	0.05	0.10	0.35	0.50

假设他观察到连续三天的显状态为：(干旱、干燥、潮湿)，然后他想预测这三天的天气状况：隐状态序列。

那么很自然，天气预报就转换为如图 4.9 所示的形式。

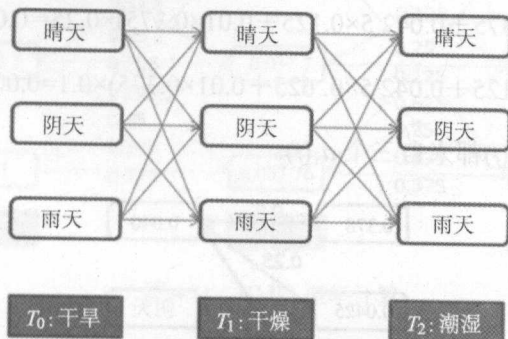


图 4.9 HMM 天气预报

根据马尔科夫链的规则，预测三天的天气状况，实际上就是计算 $T_0 \sim T_2$ 三天中所有路径中最大的一条，但现在多了一个显状态，以及该显状态对应隐状态的发射概率。前向算法的求解过程如下。

(1) 给出这三天显状态的序列：(干旱，干燥，潮湿)。为了泛化计算，也给出一般化的形式： $(b_{k1}, b_{k2}, \dots, b_{kr})$ 。

(2) 首先计算最初的状态。这一步最简单，初始状态为 Π 。初始 t_0 的显状态为干旱，则：

$$a_1(\text{晴天}) = 0.63 \times 0.6 = 0.378$$

$$a_1(\text{阴天}) = 0.17 \times 0.25 = 0.0425$$

$$a_1(\text{雨天}) = 0.20 \times 0.05 = 0.01$$

形式化上述运算如下： $\alpha_1(j) = \pi(j)b_{jk1}$

其中， b_{jkl} 是混淆矩阵，也称为发射概率， j 代表天气状态序列， k 表示空气湿度状态（观察状态）序列； π 为初始化概率向量，其初始值为 $(0.63, 0.17, 0.20)$ ，得到的 $a_1(j)$ 是第一天的状态转移概率。

然后计算最大概率的那个天气状态： $\arg \max_j (\alpha_1(j))$ 。很显然最大概率的天气状态为 $a_1(\text{晴天})$ ，最大的概率为 0.378。

(3) 接下来计算第二天的状态转移概率，因为后一天状态由前一天来决定（马尔科夫性），同时第二天的转移概率也受第二天的显状态影响。与第一天相似，使用已经得到的 $a_1(j)$ ，以及显状态—干燥的发生概率， $a_2(j)$ 的计算过程如下。

$$a_2(\text{晴天})=(0.378 \times 0.5+0.0425 \times 0.25+0.01 \times 0.25) \times 0.2=0.040425$$

$$a_2(\text{阴天})=(0.378 \times 0.375+0.0425 \times 0.125+0.01 \times 0.375) \times 0.25=0.037703125$$

$$a_2(\text{雨天})=(0.378 \times 0.125+0.0425 \times 0.625+0.01 \times 0.375) \times 0.1=0.00775625$$

注意，这里每个 $a_2(j)$ 都来自三个 $a_1(j)$ 。

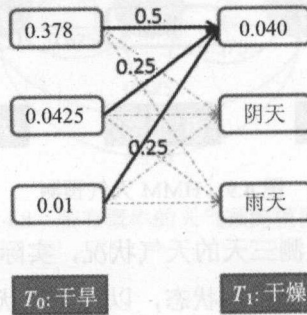


图 4.10 $a_2(\text{晴天})$ 计算

形式化上述的运算如下： $\alpha_{t+1}(j)=b_{jkt+1} \sum_{i=1}^n \alpha_i(j) \alpha_{ij}$

其中， b_{jkt+1} 是 $t+1$ 时刻的混淆矩阵，也称为发射概率， j 代表天气状态序列， k 表示空气湿度状态（观察状态）序列；得到的 $a_t(j)$ 是当前时间的状态转移概率，那么 $t+1$ 指的是下一时刻， a_{ij} 是隐状态的转移概率。

同样，计算最大概率的那个天气状态： $\arg \max_j(\alpha_{t+1}(j))$ 。同样，最大概率的天气状态为 $a_2(\text{晴天})$ ，最大的概率为 0.040425。

(4) 最后，计算第三天的状态转移概率。同样，最后一天的状态由前一天来决定，同时最后一天的转移概率也受前一天的显状态影响。与第二天的计算公式相同，使用已经得到的 $a_2(j)$ ，以及显状态—潮湿的发生概率， $a_3(j)$ 的计算过程如下。

$$a_3(\text{晴天})=(0.040425 \times 0.5+0.037703125 \times 0.25+0.00775625 \times 0.25) \times 0.05=0.0015788671875$$

$$a_3(\text{阴天})=(0.040425 \times 0.375+0.037703125 \times 0.125+0.00775625 \times 0.375) \times 0.25=0.00569521484375$$

$$a_3(\text{雨天})=(0.040425 \times 0.125+0.037703125 \times 0.625+0.00775625 \times 0.375) \times 0.5=0.0157630859375$$

运算公式与 (3) 相同： $\alpha_{t+1}(j)=b_{jkt+1} \sum_{i=1}^n \alpha_i(j) \alpha_{ij}$ 和 $\arg \max_j(\alpha_{t+1}(j))$ 。同样，最大概率的天气状态为 $a_3(\text{雨天})$ ，最大的概率为 0.0157630859375。如图 4.11 所示，得到 HMM 上的全部概率。

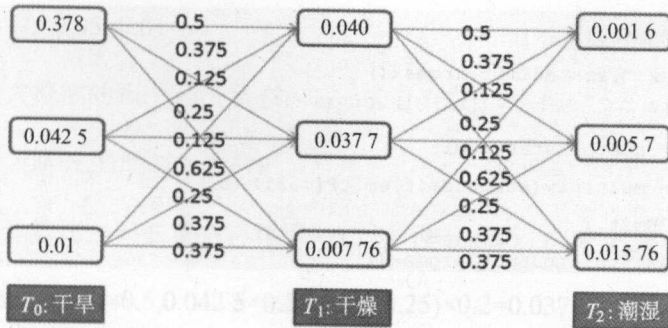


图 4.11 HMM 概率计算

(5) 根据输出概率得到每天的最大概率值。如图 4.12 所示，根据红色路径的指向，得到最终的推理结果为：晴天→晴天→雨天。

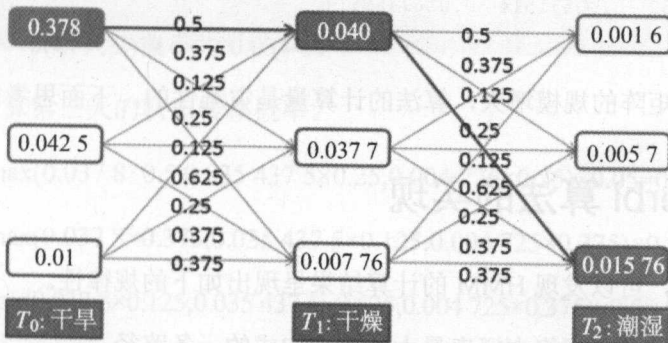


图 4.12 计算完成后的 HMM

算法代码如下。

```
# -*- coding: utf-8 -*-
# Filename : hmm.py
from numpy import *
startP = mat([0.63,0.17,0.20]) # 起始概率
stateP = mat([[0.5,0.25,0.25],[0.375,0.125,0.375],[0.125,0.675,0.375]])
# 状态转移概率
emitP = mat([[0.6,0.20,0.05],[0.25,0.25,0.25],[0.05,0.10,0.50]]) # 发射（混合）
# 概率

state1Emit = multiply(startP,emitP[:,0].T) # 计算概率：干旱—干燥—潮湿
print state1Emit
print "argmax:",state1Emit.argmax()
# 计算干燥的概率：
state2Emit = stateP*state1Emit.T
state2Emit = multiply(state2Emit,emitP[:,1])
```

```

print state2Emit.T
print "argmax:",state2Emit.argmax()
# 计算潮湿的概率:
state3Emit = stateP*state2Emit
state3Emit = multiply(state3Emit,emitP[:,2])
print state3Emit.T
print "argmax:",state3Emit.argmax()

```

结果输出如下。

```

[[ 0.378  0.0425  0.01  ]]
argmax: 0
[[ 0.040425  0.03770312  0.00796875]]
argmax: 0
[[ 0.00158152  0.00571514  0.01674551]]
argmax: 2

```

显然，随着矩阵的规模增大，算法的计算量是灾难性的。下面思考简化计算过程。

4.4.4 Viterbi 算法的实现

根据图 4.12，可以发现 HMM 的计算结果呈现出如下的规律性。

- 最优路径是有网络中概率最大的节点构成的一条路径。
- 初始状态矩阵 Π 和 t_0 时刻的转移概率决定了第一天天气状态的所有概率。
但是，对后续各天气状态产生决定影响的只有最大概率的那个天气状态：
 $\arg \max_j (\alpha_1(j))$ 。
- 之后的每一天都如此， $t+1$ 时刻的概率同时受到 t 时刻转移概率和当天的显状态影响。但是只有最大概率的那个天气状态才对后续状态产生决定性的影响：
 $\arg \max_j (\alpha_{t+1}(j))$ 。

定义 $\delta(i,t)$ 为所有序列中在 t 时刻以状态 i 终止时的最大概率。当然它所对应的那条路径就称为部分最优路径。 $\delta(i,t)$ 对于每个 (i,t) 都是存在的。这样我们就可以顺藤摸瓜找下去，在序列的最后一个状态找到整个序列的最优路径。

(1) 最初状态的计算与前面相同。

$$\delta(\text{晴天},1)=0.63\times0.6=0.378$$

$$\delta(\text{阴天},1)=0.17\times0.25=0.0425$$

$$\delta(\text{雨天}, 1) = 0.20 \times 0.05 = 0.01$$

然后计算最大概率的那个天气状态: $\arg \max_i (\delta(i, t))$ 。

计算结果为 $\delta(\text{晴天}, 1)$ 概率为 0.378。

(2) 接下来计算第二天的状态转移概率: $\delta_t(i) = \max_j (\delta_{t-1}(j) \alpha_{ji} b_{ikt})$ 。

$$\delta(\text{晴天}, 2) = \max(0.378 \times 0.5, 0.0425 \times 0.25, 0.01 \times 0.25) \times 0.2 = 0.0378$$

$$\delta(\text{阴天}, 2) = \max(0.378 \times 0.375, 0.0425 \times 0.125, 0.01 \times 0.375) \times 0.25 = 0.0354375$$

$$\delta(\text{雨天}, 2) = \max(0.378 \times 0.125, 0.0425 \times 0.625, 0.01 \times 0.375) \times 0.1 = 0.004725$$

同样, 计算最大概率的那个天气状态: $\arg \max_i (\delta_i(i, t))$

计算结果为, $\delta(\text{晴天}, 2)$ 概率为 0.0378。

(3) 最后计算第三天的状态转移概率。

$$\delta(\text{晴天}, 3) = \max(0.0378 \times 0.5, 0.0354375 \times 0.25, 0.004725 \times 0.25) \times 0.05 = 0.000945$$

$$\delta(\text{阴天}, 3) = \max(0.0378 \times 0.375, 0.0354375 \times 0.125, 0.004725 \times 0.375) \times 0.25 = 0.00354375$$

$$\delta(\text{雨天}, 3) = \max(0.0378 \times 0.125, 0.0354375 \times 0.625, 0.004725 \times 0.375) \times 0.5 = 0.01107421875$$

最后, 计算最大概率的那个天气状态为, $\delta(\text{雨天}, 3)$ 概率为 0.01107421875。

根据计算结果, 形成最优路径, 与 4.4.3 节的算法相同, 如图 4.13 所示。

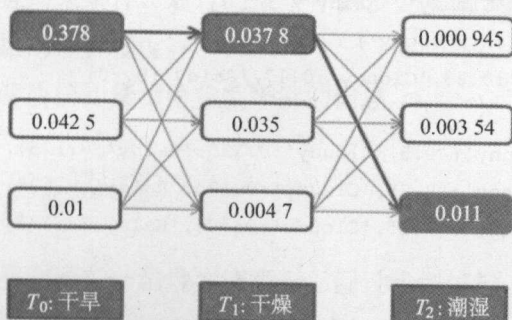


图 4.13 Viterbi 计算结果

(1) Viterbi 算法的 Python 实现, 代码如下。

```
def viterbi(obs, states, start_p, trans_p, emit_p):
    """
    :obs: 观测序列
```



```

:states: 隐状态
:start_p: 初始概率 (隐状态)
:trans_p: 转移概率 (隐状态)
:emit_p: 发射概率 (隐状态表现为显状态的概率)
"""
V = [{}] # 路径概率表 V[时间][隐状态] = 概率
for y in states: # 初始化初始状态 (t == 0)
    V[0][y] = start_p[y] * emit_p[y][obs[0]]
for t in xrange(1, len(obs)): # 对 t > 0 跑一遍维特比算法
    V.append({})
    for y in states:
        # 概率 隐状态 = 前状态是 y0 的概率 * y0 转移到 y 的概率 * y 表现为当前状态的概率
        V[t][y] = max([(V[t-1][y0] * trans_p[y0][y] * emit_p[y][obs[t]]) for
y0 in states])
    result = []
    for vector in V:
        temp={}
        temp[vector.keys()[argmax(vector.values())]]=max(vector.values())
        result.append(temp)
    return result

```

上述为 Viterbi 算法的 Python 实现。下面给出函数的输出结果。

根据上述案例提供的数据，执行 Viterbi 算法主程序，代码如下。

```

# -*- coding: UTF-8 -*-
from numpy import *

states = ('Sunny','Cloudy','Rainy')
obs = ('dry','dryish','soggy')
start_p = {'Sunny':0.63,'Cloudy':0.17,'Rainy':0.20}
trans_p = {
    'Sunny' : {'Sunny': 0.5,'Cloudy':0.375,'Rainy':0.125},
    'Cloudy': {'Sunny': 0.25,'Cloudy':0.125,'Rainy':0.625},
    'Rainy' : {'Sunny': 0.25,'Cloudy':0.375,'Rainy':0.375},
}

emit_p = {
    'Sunny' : {'dry':0.60,'dryish':0.20,'soggy':0.05},
    'Cloudy': {'dry':0.25,'dryish':0.25,'soggy':0.25},
    'Rainy' : {'dry':0.05,'dryish':0.10,'soggy':0.50},
}

print viterbi(obs,states, start_p, trans_p, emit_p)

```

执行结果如下。

```
[{'Sunny': 0.378}, {'Sunny': 0.0378}, {'Rainy': 0.01107421875}]
```

与手工计算结果相同。

4.5 最大熵模型

隐马尔科夫模型（以下称 HMM）将标注问题看作一个马尔科夫链，针对相邻标注的关系进行建模，其中每个标记都对应一个概率。HMM 在概率图模型中属于产生式模型，需要依赖一个联合概率分布。为了得到这个联合概率分布，产生式模型需要枚举出所有可能的观察序列，这在 NLP 的实际运算过程中往往是很困难的。

另一方面，NLP 的大量真实语料中，观测序列更多地是以一种多重的交互特征形式表现，观测序列之间广泛存在长程相关性。例如，在命名实体识别任务中，由于专名结构所具有的复杂性，利用简单的特征函数往往无法涵盖所有的特性（它无法使用多于一个标记的特征）。

此时，需要引入一个新的模型。由于该模型获得的是所有满足约束条件的模型中信息熵极大的标签，因此称为最大熵模型。最大熵模型有如下特点。

- ☐ 可以使用任意的复杂相关特征。
- ☐ 可以灵活地设置约束条件，通过约束条件的多少来调节模型对未知数据的适应度和对已知数据的拟合程度。
- ☐ 它还能自然地解决统计模型中参数平滑的问题。

4.5.1 从词性标注谈起

下面通过一个简单的例子来介绍最大熵概念。假设我们模拟一个汉语中词性标注的过程。汉语中很多词有多个标签，如“把”就有如下 5 个词性标签。

- ☐ 数词：个把月。
- ☐ 名词：车把、门把。
- ☐ 介词：把门关上。
- ☐ 量词：一把伞、一把锁。

□ 动词：把门、把住。

我们为“把”的每个词性分配一个估计值 p (词性)，即系统选择模型 p 作为某个词性出现的概率。为了得到模型 p ，我们收集大量的语料样本。我们的目标有两个：一个是从样本中抽取一组决策过程的事实（规则）；另一个是基于这些事实构建这一词性标注模型。

为了简便，我们把“把”的 5 种词性写成集合的形式： $\{\text{数词、名词、介词、量词、动词}\}$ 。这样，就产生了模型的一个约束条件，内容如下。

$$p(\text{介词})+p(\text{动词})+p(\text{量词})+p(\text{名词})+p(\text{数词})=1$$

上述等式代表了“把”字各种词性出现概率的统计公式，即不论 $p(\text{词性})$ 取任何值，其总和必须为 1。

显然， $p(\text{词性})$ 可以取无数个数，那么就有无数种满足这个条件的组合可供选择。假设一种极端的情况是 $p(\text{介词})=1$ ，这个模型总是预测为介词，另一个满足这一约束的模型是 $p(\text{介词})=1/2$ 和 $p(\text{量词})=1/2$ 。当然，这两个模型都有违常理：但是，我们仅知道“把”共有这 5 个词性，而它们的实际概率分布我们一无所知。在没有任何语料支持的情况下，我们只能使用熵最大的均匀分布（uniform），这就是所谓的最大熵原则，内容如下。

$$p(\text{介词}) = 1/5$$

$$p(\text{动词}) = 1/5$$

$$p(\text{量词}) = 1/5$$

$$p(\text{名词}) = 1/5$$

$$p(\text{数词}) = 1/5$$

这个模型将概率均匀分配给 5 个可能的词性。这毕竟不是我们满意的结果。为此，引入一定数量的语料样本。在样本中有 30% 的比例，“把”被标注为“介词”和“量词”。这样，可以使用这些经验的数据更新模型的约束条件，内容如下。

$$p(\text{介词})+p(\text{量词})=3/10$$

$$p(\text{介词})+p(\text{动词})+p(\text{量词})+p(\text{名词})+p(\text{数词})=1$$

那么，如何确定剩下几个词性的概率呢？很简单，在没有其他知识的情况下，按照最大熵原则的合理模型 p 是均匀分布，也就是在满足约束的条件下，将概率尽可能均匀地分配。均匀分布示例如表 4.9 所示。

表 4.9 均匀分布示例

已知的约束条件	未知的均匀分布（最大熵原则）
$p(\text{介词})=3/20$ $p(\text{量词})=3/20$	$p(\text{动词})=7/30$ $p(\text{名词})=7/30$ $p(\text{数词})=7/30$

继续引入语料样本，随着语料数量的增大，我们发现“把”作为介词和名词的概率显著增加，单独记录下这个统计结果，把它作为统计信息的第三个约束，内容如下。

$p(\text{介词})+p(\text{量词})=3/10$

$p(\text{介词})+p(\text{动词})+p(\text{量词})+p(\text{名词})+p(\text{数词})=1$

$p(\text{量词})+p(\text{名词})=1/2$

我们可以再次寻找满足这些约束的均匀分配的模型 p 。当然，可以像表 4.9 一样很简单地构建一种新表。但这一次的结果没有那么明显。由于增加了问题的复杂度，现在出现了如下两个问题。首先，“均匀”究竟是什么意思呢？如何度量一个模型的均匀度（uniformity）？其次，有了这些问题的答案之后，如何找到满足一组约束且最均匀的模式？

最大熵的方法回答了上述两个问题。直观上讲，很简单，即根据已知的知识进行建模，而对未知的不做任何假设（model all that is known and assume nothing about that which is unknown）。换句话说，在给定一组事实（features+output）的条件下，选择符合所有事实，并在未知方面使用熵最大的均匀模型。这恰恰是上述例子中每一步选择模型 p 所采取的方法。

4.5.2 特征和约束

根据上例构造一个概率图模型，它产生一个输出 y ， y 属于一个有穷集合 \mathbf{Y} 。对于刚才词性标注的例子，该过程输出词汇“把”的词性，输出值 y 可以是集合 {数词、名词、介词、量词、动词} 中任何一个标签。在输出 y 时，该过程可能会受上下文信息的影响，我们把这个语境因素定义为 x ， x 属于有穷的集合 \mathbf{X} 。在词性标注的例子中， \mathbf{X} 是指句子中出现在“把”周围的所有词汇。而我们的任务是构造一个统计模型，该模型的任务是预测在给定上下文 x 的情况下，输出 y 的概率： $p(y|x)$ 。

为了得到这个模型，我们需要收集大量的语料样本： $(x_1,y_1), (x_2,y_2), \cdots, (x_n,y_n)$ 。这里每一个样本都包含“把”的上下文词汇 x ，以及“把”对应的标签 y 。这需要大量的

文本语料, 有关语料的收集、整理和处理, 在后面的章节将有详细的介绍, 本章暂不涉及。

但不论你收集多少语料, 它们都不是语言文本的总体(语言的生成具有动态性), 而只是样本, 换句话说不能用 p 来表示, 于是引入一个新的经验分布函数 \tilde{p} 来表达训练样本:

$$\tilde{p}(x,y) \equiv \frac{1}{N}(x,y) \quad \text{在样本中出现的次数}$$

通常, 对于一个特定的 (x_i, y_i) 对, 它要么不出现在样本中, 要么最多出现 N 次。

我们的目标是根据收集到的语料样本构建 $\tilde{p}(x,y)$ 的概率模型。构成该概率模型的组件将是一组训练样本的统计值。在上例中, 已经采用了几个统计数据: (1) $p(\text{介词})+p(\text{量词})=3/10$ 。(2) $p(\text{量词})+p(\text{名词})=1/2$, 等等。这些统计模式都是上下文独立的, 但也可以考虑上下文依赖信息 x 的模式。例如, 在语料样本中, 如果“一”出现在“把”之前, 那么“把”属于“量词”的概率为 $9/10$ 。

为了表示这个事件(event), 即当“一”出现在“把”之前, “把”被标注为“量词”, 引入了如下指示函数(取值为 0、1 两种情况):

$$f(x,y) = \begin{cases} 1 & \text{if } y=\text{量词} \text{ 并且 “一” 出现在 “把” 之前} \\ 0 & \text{otherwise} \end{cases}$$

其中, 特征 f 为关于经验分布 $\tilde{p}(x,y)$ 的期望值。将这个期望值表示为:

$$\tilde{P}(f_i) = \sum_{x,y} \tilde{p}(x,y) f_i(x,y) \quad (4.5.1)$$

可以将任何样本的统计量表示成一个适当的二值指示函数的期望值, 这个函数叫作特征函数 (Feature Function) 或简称为特征 (Feature)。

概率模型的一个设计要点就是要找到符合要求的统计量。该统计量要能使模型通过拟合达到希望的结果。这个拟合过程通过约束模型 p 分配给相应特征函数的期望值来实现。特征 f 关于模型 $p(y|x)$ 的期望值:

$$P(f_i) = \sum_{x,y} p(x)p(y|x) f_i(x,y) \quad (4.5.2)$$

这里, $\tilde{p}(x)$ 是来自语料样本中的 x 的经验分布。约束这一期望值和语料样本中 f 的期望值相同。

这里隐含一个假设,即样本的分布与总体的分布关系,根据大数定理,样本要足够大才能够满足如下要求:

$$P(f_i) \equiv \tilde{P}(f_i) \quad (4.5.3)$$

组合上述的式(4.5.1)、式(4.5.2)和式(4.5.3),得到如下等式:

$$\sum_{x,y} \tilde{p}(x,y) f(x,y) = \sum_{x,y} \tilde{p}(x) p(y|x) f(x,y)$$

式(4.5.3)称为约束等式(Constraint Equation)或者简称为约束(Constraint)。我们只关注那些满足式(4.5.3)的概率模型 $p(y|x)$,而不考虑那些和训练样本中特征 f 频率不一致的模型。

最后,关于特征和约束,这里再强调两句:“特征”和“约束”在讨论最大熵时经常被混用,我们希望读者注意区分这两者的概念:特征(Feature)是 (x,y) 的二值函数;约束是一个等式,即模型的特征函数期望值等于训练样本中特征函数的期望值。

4.5.3 最大熵原理

假设有 n 个特征函数 f_i , 它们的期望值决定了在建模过程中重要的统计数据。要想模型符合这些统计,也就是说,要想模型 E 属于 P 的子集 C , 即

$$C \equiv \left\{ p \in P \mid p(f_i) = \tilde{p}(f_i) \text{ for } i \in \{1, 2, \dots, n\} \right\} \quad (4.5.4)$$

图 4.14 所示为式(4.5.4)约束的几何解释。这里, P 是所有可能的概率分布空间。如果不施加任何约束,如图 4.14 (a) 所示,所有概率模型都是允许的。施加第一个线性约束 C_1 后,模型被限制在 C_1 定义的区域,如图 4.14 (b) 所示。如果施加两个约束且第二个线性约束如图 4.14 (c) 所示,则可以准确地确定 p 。另一种情形是,第二个线性约束与第一个不一致,例如,第一个约束可能需要的概率是 $1/3$,第二个约束需要的概率是 $3/4$,如图 4.14 (d) 所示。在目前的设置中,线性约束是从语料数据集中抽取的,数据量庞大不可能手工构造,在众多的数据中不一致的可能性很小。进一步来说,在应用中的线性约束甚至不会接近唯一确定的 p ,就像图 4.14 (c) 所示的那样。相反,集合 $C = C_1 \cap C_2 \cap \dots \cap C_n$ 中的模型是无穷的。

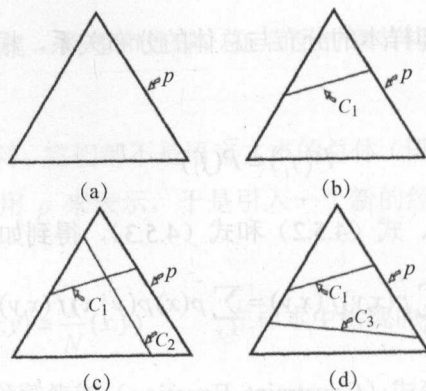


图 4.14 最大熵模型

属于集合 C 的所有模型 p 中，最大熵的理念决定我们选择最均匀的分布。但现在，我们面临一个前面遗留的问题：什么是“均匀分布”？数学上，条件分布 $p(y|x)$ 的均匀度被定义为条件熵：

$$H(p) = -\sum_{x,y} \tilde{p}(x)p(y|x) \log p(y|x) \quad (4.5.5)$$

熵的下界是 0，这时模型没有任何不确定性；熵的上界是 $\log|Y|$ ，即在所有可能 ($|Y|$ 个) 的 y 上均匀分布。有了这个定义，我们就可以提出最大熵原则。

当从允许的概率分布集合 C 中选择一个模型时，选择模型 $p^* \in C$ ，使得熵 $H(p)$ 最大。

$$p^* = \arg \max_{p \in C} H(p) \quad (4.5.6)$$

这是之前所称的原始问题的展开。简单地讲，我们的目标是在 $p \in C$ 的情况下，最大化 $H(p)$ 。

该约束问题的完整形式如下：

$$\begin{aligned} \text{find } p^* &= \arg \max_{p \in C} H(p) \\ &= \arg \max_{p \in C} \left(-\sum_{x,y} \tilde{p}(x)p(y|x) \log p(y|x) \right) \end{aligned}$$

目标函数的约束条件如下：

- (1) $p(y|x) \geq 0$ ，对于所有的 x, y 。
- (2) $\sum_y p(y|x) = 1$ ，对于所有的 x 。确保 p 是一个条件概率。
- (3) $\sum_{x,y} \tilde{p}(x,y)f(x,y) = \sum_{x,y} \tilde{p}(x)p(y|x)f(x,y)$ 。

其中, $i \in \{1, 2, \dots, n\}$ 。换句话说, $p \in C$, 满足有效约束 C 。

4.5.4 公式推导

为了解决优化问题, 引入拉格朗日乘子:

$$\begin{aligned} \xi(p, A, \gamma) = & -\sum_{x,y} \tilde{p}(x) p(y|x) \log p(y|x) \\ & + \sum_i \lambda_i \left(\sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) - \sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right) \\ & + \gamma \left(\sum_x p(y|x) - 1 \right) \end{aligned} \quad (4.5.7)$$

实值参数 γ 和 $A = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ 对应施加在解上的 $n+1$ 个约束。下面的策略可以求出 p 的最优解 (p^*)。

首先, 将 γ 和 $A = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ 看成常量, 寻找 p 最大化公式, 即式 (4.5.7)。这会产生以 γ 和 $A = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ 为参数的表示式 p (参数没有解决)。接着, 将该表达式代回式 (4.5.7) 中, 这次求 γ 和 $A = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ 的最优解 (γ^* 和 A^*)。

按照这一方式, 固定 γ 和 A , 计算在所有 $p \in P$ 空间下 $\xi(p, A, \gamma)$ 的无约束的最大值。

$$\begin{aligned} \frac{\partial \xi}{\partial p(y|x)} = & -\tilde{p}(x) \left(\log p(y|x) + \frac{p(y|x)}{p(y|x)} \right) + \sum_i \lambda_i \tilde{p}(x) f_i(x,y) + \gamma \\ = & -\tilde{p}(x) (\log p(y|x) + 1) + \sum_i \lambda_i \tilde{p}(x) f_i(x,y) + \gamma \end{aligned}$$

令该式等于 0, 求解 $p(y|x)$:

$$\begin{aligned} -\tilde{p}(x) (\log p(y|x) + 1) + \sum_i \lambda_i \tilde{p}(x) f_i(x,y) + \gamma &= 0 \\ \tilde{p}(x) (\log p(y|x) + 1) &= \sum_i \lambda_i \tilde{p}(x) f_i(x,y) + \gamma \\ \log p(y|x) + 1 &= \sum_i \lambda_i f_i(x,y) + \frac{\gamma}{\tilde{p}(x)} \\ \log p(y|x) &= \sum_i \lambda_i f_i(x,y) + \frac{\gamma}{\tilde{p}(x)} - 1 \\ p(y|x) &= \exp \left(\sum_i \lambda_i f_i(x,y) \right) \exp \left(\frac{\gamma}{\tilde{p}(x)} - 1 \right) \end{aligned} \quad (4.5.8)$$

可以看出式 (4.5.8) 的第二个因子对应第二个约束: $\sum_y p(y|x)=1$ 。

将上式带入式 (4.5.8) 得到:

$$1 = \sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right) \exp\left(\frac{\gamma}{p(x)} - 1\right) \exp\left(\frac{r}{p(x)} - 1\right) = \frac{1}{\sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right)} \quad (4.5.9)$$

将式 (4.5.9) 带入式 (4.5.8), 得到:

$$p(y|x) = \exp\left(\sum_i \lambda_i f_i(x, y)\right) \cdot \frac{1}{\sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right)} \quad (4.5.10)$$

因此得到:

$$p^*(y|x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right) \quad (4.5.11)$$

$Z(x)$ 是正则化因子。

4.5.5 对偶问题的极大似然估计

现在要求解最优值 γ^* 和 λ^* 。显然, 已经知道了 γ^* , 还不知道 λ^* 。为此, 引入对偶函数 Ψ :

$$\Psi(\lambda) \equiv \xi(p^*, \lambda, \gamma^*) \quad (4.5.12)$$

对偶优化问题如下:

$$\text{find } \lambda^* = \arg\max_{\lambda} \Psi(\lambda) \quad (4.5.13)$$

因为 p^* 和 γ^* 是固定的, 式 (4.5.13) 的右边只有自由变量 $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ 。

参数值等于 $\lambda = \lambda^*$ 的 p^* 就是一开始约束优化问题的最优解。这是拉格朗日乘子理论的基本原理, 通常叫作 Kuhn-Tucker Theorem (KTT)。最后结论如下。

满足约束 C 最大熵模型具有式 (4.5.13) 的参数化形式, 最优参数 λ^* 可以通过最小化对偶函数 $\Psi(\lambda)$ 求得。

补充说明如下。

$\Psi(\lambda)$ 究竟是什么样呢？记住，我们要求 $\xi(p^*, \lambda, \gamma^*)$ 的最小值，这是拉格朗日乘子理论的基本原理。

$$\begin{aligned}
 \Psi(\lambda) &= \xi(p^*, \lambda, \gamma^*) = -\sum_{x,y} \tilde{p}(x)p(y|x) \log p(y|x) \\
 &\quad + \sum_i \lambda_i \left(\sum_{x,y} \tilde{p}(x)p(y|x) f_i(x,y) - \sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right) \\
 &= -\sum_{x,y} \tilde{p}(x)p(y|x) \log \left(\frac{\exp \left(\sum_i \lambda_i f_i(x,y) \right)}{Z_\lambda(x)} \right) + \sum_{x,y} \left(\tilde{p}(x)p(y|x) \sum_i \lambda_i f_i(x,y) \right) - \sum_i \lambda_i \left(\sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right) \\
 &= -\sum_{x,y} \left(\tilde{p}(x)p(y|x) \sum_i \lambda_i f_i(x,y) \right) + \sum_x \tilde{p}(x) \left(\sum_y p(y|x) \right) \log Z_\lambda(x) \\
 &\quad + \sum_{x,y} \left(\tilde{p}(x)p(y|x) \sum_i \lambda_i f_i(x,y) \right) - \sum_i \lambda_i \left(\sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right) \\
 &= \sum_x \tilde{p}(x) \log(Z_\lambda(x)) - \sum_i \lambda_i \left(\sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right) \\
 &= \sum_x \tilde{p}(x) \log(Z_\lambda(x)) - \sum_i \lambda_i \tilde{p}(f_i)
 \end{aligned}$$

在目前情况下，要说出最终的结果是很容易的：假设 λ^* 是对偶问题的解， p_{λ^*} 是原问题的解。那么， $p_{\lambda^*} = p^*$ 。换句话说，最大熵模型的约束条件 C 有一个形式参数 p_{λ^*} ，其中，参数值 λ^* 可以通过最大化对偶函数 $\Psi(\lambda)$ 来确定。表 4.10 总结了我们已经建立的原问题—对偶问题的框架。

表 4.10 最大熵的对偶问题

最大熵和最大似然度额度对偶性是一个约束规划中最一般的对偶现象		
	原问题	对偶问题
解决方案的问题 描述类型	$\arg \max_{p \in C} H(p)$	$\arg \max_{\lambda} \Psi(\lambda)$
	最大熵约束优化: $p \in C$ p^* 熵函数是一个凸函数	最大似然无约束优化: 实值向量 $\{\lambda_1, \lambda_2, \dots, \lambda_s\}$ 凸函数的极值与其对偶问题的极值相同
Kuhn-Tucker 定理: $p^* = p_{\lambda^*}$		

下面使用经验分布 \tilde{p} 的极大似然法的对数似然度 $L_{\tilde{p}}(p)$, 作为模型 p 的预测, 定义为

$$L_{\tilde{p}}(p) \equiv \log \prod_{x,y} p(y|x)^{\tilde{p}(x,y)} = \sum_{x,y} \tilde{p}(x,y) \log p(y|x) \quad (4.5.14)$$

推导过程:

$$\begin{aligned} \sum_{x,y} \tilde{p}(x,y) \log p(y|x) &= \sum_{x,y} \tilde{p}(x,y) \log \exp \left(\sum_i \lambda_i f_i(x,y) \right) - \sum_{x,y} \tilde{p}(x,y) \log Z_{\lambda}(x) \\ &= \sum_i \lambda_i \left(\sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right) - \sum_{x,y} \tilde{p}(x,y) \log Z_{\lambda}(x) \\ &= \sum_i \lambda_i \tilde{p}(f_i) - \sum_{x,y} \tilde{p}(x,y) \log Z_{\lambda}(x) \\ &= -\psi(\lambda) \end{aligned}$$

因此, 有: $\max(L_{\tilde{p}}(p)) = \min(\psi(\lambda))$

这里的 p 具有式 (4.5.14) 的形式, 其结果进一步可以表述为: 最大熵模型 $p^* \in C$ 是具有式 (4.5.14) 的形式, 且最大化样本似然率的模型。最大熵模型与其极大似然估计的模型一致。

对偶函数 $\Psi(\lambda)$ 其实就是指数模型 p_{λ} 的对数似然度, 即

$$\Psi(\lambda) = L_{\tilde{p}}(p_{\lambda})$$

按照这种解释, 前面所述的结果可以改写为:

含有最大熵的模型 $p \in C$ 就是参数族为 $P_{\lambda}(y|x)$ 的模型, 它最大化了与训练样本 \tilde{P} 的似然度。

4.5.6 GIS 实现

要计算 λ , 直接求解解析解肯定是行不通的。对于最大熵模型对应的最优化问题, GIS、lbfgs、sgd 等最优化算法都能解。相比之下, GIS 大概是最好实现的。这里只介绍 GIS 算法。具体步骤如下:

(1) 设置 $\lambda_i^{(0)}$ 等于任意值, 如等于 0。

$$\lambda_i^{(0)} = 0$$

(2) 重复直到收敛。

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \frac{1}{M} \log \frac{E_{\bar{p}} f_i}{E_{p^{(0)}} f_i}$$

这里, (t) 是迭代下标, 常数 M 定义为: $M = \max_{x,y} \sum_{i=1}^n f_i(x,y)$

实践中 M 是训练样本中最大的特征个数。 M 再大些也没关系, 但是它决定了收敛速度, 还是取最小的好。实际上, GIS 算法用第 N 次迭代的模型来估算每个特征在训练数据中的分布。如果超过了实际的, 就把相应参数 λ 变小。否则, 将它们变大。当训练样本的特征分布和模型的特征分布相同时, 就求得了最优参数。

案例代码如下。

```
# -*- coding: utf-8 -*-
import sys
import os
from collections import defaultdict
import math

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

class MaxEnt(object):
    def __init__(self):
        self.feats = defaultdict(int)
        self.trainset = [] # 训练集
        self.labels = set() # 标签集

    def load_data(self, file):
        for line in open(file):
            fields = line.strip().split()
            if len(fields) < 2: continue # 特征数要大于两列
            label = fields[0] # 默认第一列是标签
            self.labels.add(label)
            for f in set(fields[1:]):
                self.feats[(label,f)] += 1 # (label,f) 元组是特征 # print label,f
            self.trainset.append(fields)

    def _initparams(self): # 初始化参数
```



```

self.size = len(self.trainset)
self.M = max([len(record)-1 for record in self.trainset]) # GIS 训练算法
# 的 M 参数

self.ep_ = [0.0]*len(self.feats)
for i,f in enumerate(self.feats):
    self.ep_[i] = float(self.feats[f])/float(self.size) # 计算经验分布的
# 特征期望

    self.feats[f] = i # 为每个特征函数分配 id
self.w = [0.0]*len(self.feats) # 初始化权重
self.lastw = self.w

def probwgt(self, features, label): # 计算每个特征权重的指数
    wgt = 0.0
    for f in features:
        if (label,f) in self.feats:
            wgt += self.w[self.feats[(label,f)]]
    return math.exp(wgt)

"""
calculate feature expectation on model distribution
"""
def Ep(self): # 特征函数
    ep = [0.0]*len(self.feats)
    for record in self.trainset: # 从训练集中迭代输出特征
        features = record[1:]
        prob = self.calprob(features) # 计算条件概率  $p(y|x)$ 
        for f in features:
            for w,l in prob:
                if (l,f) in self.feats: # 来自训练数据的特征
                    idx = self.feats[(l,f)] # 获取特征 id
                    ep[idx] += w * (1.0/self.size) #  $\sum (1/N * f(y,x) * p(y|x))$ ,
p(x) = 1/N
    return ep

def _convergence(self, lastw, w): # 收敛—终止条件
    for w1,w2 in zip(lastw,w):
        if abs(w1-w2) >= 0.01: return False
    return True

def train(self, max_iter =1000): # 训练样本的主函数。默认迭代次数 1000 次
    self._initparams() # 初始化参数
    for i in range(max_iter):

```

```

print 'iter %d ...'% (i+1)
self.ep = self.Ep() # 计算模型分布的特征期望
self.lastw = self.w[:]
for i,win in enumerate(self.w):
    delta = 1.0/self.M * math.log(self.ep_[i]/self.ep[i])
    self.w[i] += delta # 更新 w
print self.w , self.feats
if self._convergence(self.lastw,self.w): # 判断算法是否收敛
    break

def calprob(self,features): #计算条件概率
    wgts = [(self.probwgt(features, l),l) for l in self.labels] #
    Z = sum([ w for w,l in wgts]) # 归一化参数
    prob = [ (w/Z,l) for w,l in wgts] # 概率向量
    return prob

def predict(self,input): # 预测函数
    features = input.strip().split()
    prob = self.calprob(features)
    prob.sort(reverse=True)
    return prob

```

数据文件如下。

训练集	整理后的特征:
Outdoor Sunny Happy	Outdoor Sunny 5
Outdoor Sunny Happy Dry	Outdoor Happy 5
Outdoor Sunny Happy Humid	Outdoor Dry 2
Outdoor Sunny Sad Dry	Outdoor Humid 6
Outdoor Sunny Sad Humid	Outdoor Sad 4
Outdoor Cloudy Happy Humid	Outdoor Cloudy 4
Outdoor Cloudy Happy Humid	Indoor Rainy 4
Outdoor Cloudy Sad Humid	Indoor Humid 4
Outdoor Cloudy Sad Humid	Indoor Happy 2
Indoor Rainy Happy Humid	Indoor Dry 2
Indoor Rainy Happy Dry	Indoor Sad 4
Indoor Rainy Sad Dry	Indoor Cloudy 2
Indoor Rainy Sad Humid	
Indoor Cloudy Sad Humid	
Indoor Cloudy Sad Humid	

测试代码部分如下。

```
# -*- coding: utf-8 -*-
import sys
import os
import maxent

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

model = maxent.MaxEnt()
model.load_data('data.txt') # 导入训练集
model.train() # 训练模型
print model.predict("Rainy Happy Dry") # 预测结果
```

训练结果如下。

```
iter 1 ...
[0.0, 0.23104906018664842, 0.0, -0.18653859597847416, 0.23104906018664842, 0.0,
0.11889164797957746, -0.13515503603605475, 0.0, -0.07438118377140324, 0.060773852264651596,
0.09589402415059367]
defaultdict(<type 'int'>, {'Outdoor', 'Sad'): 0, ('Outdoor', 'Sunny'): 1,
('Outdoor', 'Dry'): 2, ('Indoor', 'Happy'): 3, ('Indoor', 'Rainy'): 4, ('Indoor',
'Dry'): 5, ('Outdoor', 'Happy'): 6, ('Indoor', 'Cloudy'): 7, ('Indoor', 'Sad'): 8,
('Indoor', 'Humid'): 9, ('Outdoor', 'Humid'): 10, ('Outdoor', 'Cloudy'): 11})
iter 2 ...
...
iter 144 ...
[-0.3644512232512526, 3.7191730818297613, -0.05544928580348433, -1.3399214304769507,
4.703394433851156, 0.058616584006608345, 0.6051416833043833, -0.584974488792124,
0.37383730234345736, 0.04635901193875606, -0.018818546603234308, 0.3283161400233158]
defaultdict(<type 'int'>, {'Outdoor', 'Sad'): 0, ('Outdoor', 'Sunny'): 1,
('Outdoor', 'Dry'): 2, ('Indoor', 'Happy'): 3, ('Indoor', 'Rainy'): 4, ('Indoor',
'Dry'): 5, ('Outdoor', 'Happy'): 6, ('Indoor', 'Cloudy'): 7, ('Indoor', 'Sad'): 8,
('Indoor', 'Humid'): 9, ('Outdoor', 'Humid'): 10, ('Outdoor', 'Cloudy'): 11})
```

预测结果如下。

```
[(0.9464649414197988, 'Indoor'), (0.05353505858020124, 'Outdoor')]
```


4.6 条件随机场模型

条件随机场 (Conditional Random Fields, CRF) 最早由 Lafferty 等人于 2001 年提出, 其模型思想的主要来源是隐马尔科夫模型。

对 HMM 模型的三个基本问题的解决用到了 HMM 模型中提到的方法如 Forward-Backward 算法和 Viterbi 算法。可以把条件随机场看成一个无向图模型或马尔科夫随机场, 它是一种用来标记和切分序列化数据的统计模型。该模型是在给定需要标记的观察序列的条件下, 计算整个标记序列的联合概率, 而不是在给定当前状态的条件下, 定义下一个状态的分布。标记序列 (Label Sequence) 的分布条件属性, 可以让 CRF 很好地拟和现实数据, 而在这些数据中, 标记序列的条件概率信赖于观察序列中非独立的、相互作用的特征, 并通过赋予特征以不同权值来表示特征的重要程度。

下文说明了条件随机场的思想和理论基础。首先, 给出条件随机场的通用公式, 接着深入讨论 CRF 最普遍的形式, 具有线性序列的结构。主要的重点是训练和推理方面。本节以简短的内容讨论了任意结构的 CRF。

4.6.1 随机场

随机场是一个随机过程的推广, 使得底层的参数不再是一个简单的实数或整型“时间”, 而是一个多维向量或点。在离散的情况下, 随机场可以被认为是一组被确定在某个空间 (如 n 维欧几里得空间) 的离散点 (随机数) 列表。在统计学中, 随机场可以看成一组随机变量的集合, 而这组随机变量对应同一个样本空间。当然, 这些随机变量之间常有某种依赖关系, 一般来说, 也只有当这些变量之间有依赖关系的时候, 我们将其单独拿出来看成一个随机场才有实际意义。

因此, 确定一个随机场有如下重要的要素。第一, 随机过程包含了哪些不同的样本空间; 第二, 样本空间中的随机变量相互之间形成何种依赖关系 (换个角度讲, 就是条件独立的关系)。这两个因素是构成“随机场”的基本要件。

随机场包含如下两个要素: 样本空间集合和单个样本空间中的随机变量集合。

我们不妨拿种地来打个比方。“不同分布的样本空间”好比一亩亩农田; “相互依赖的随机变量”好比种的各种庄稼。我们可以给不同的地种上不同的庄稼, 这就好比给随机场的每个“样本空间”里赋予不同取值范围的随机变量。通俗一点地讲, 随机场就是在哪块地里种什么庄稼的事情。

随机场 (Random Field) 定义如下: 在概率论中, 由样本空间 $\Omega = \{0, 1, \dots, G-1\}^n$ 取样构成的随机变量 X_i 所组成的 $S = \{X_1, \dots, X_n\}$ 。若对所有的 $\omega \in \Omega$, $\pi(\omega) > 0$ 。则称 π 为一个随机场。一些已有的随机场, 如吉布斯随机场 (GRF)、马尔科夫随机场 (MRF)、条件随机场 (也常被称为马尔科夫随机场, CRF) 和高斯随机场。

所以, 求解一个随机场, 就是要找到有多少种不同的样本空间, 以及其中随机变量的概率分布。

4.6.2 无向图的团 (Clique) 与因子分解

引入团的定义: 无向图 G 的某个子图 S , 若 S 中的任何两个节点均有边, 则 S 称为 G 的团 (Clique)。

若 C 是 G 的一个团, 并且不能再加入任何一个 G 的节点使其称为团, 则称 C 为 G 的最大团 (Maximal Clique)。举例如下。

图 4.15 中的团包括如下内容。

$\{1,2\}, \{1,3\}, \{2,3\}, \{2,4\}, \{3,4\}, \{3,5\}, \{1,2,3\}, \{2,3,4\}$ 。

其中, 最大团包括: $\{1,2,3\}, \{2,3,4\}, \{3,5\}$ 。

对于一个无向图而言, 其联合分布可以表示成最大团上的随机变量函数的乘积形式, 这个操作称为无向图的因子分解 (Factorization)。

如图 4.15 所示, 其因子分解可以表示为如下形式。

$$p(y|\theta) = \frac{1}{Z(\theta)} \psi_{123}(y_1, y_2, y_3) \psi_{234}(y_2, y_3, y_4) \psi_{35}(y_3, y_5)$$

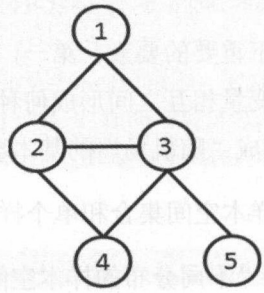


图 4.15 无向图的团

因为, $p(y|\theta)$ 是一个取值为 $0 \sim 1$ 的概率, $Z(\theta)$ 为归一化因子。

将其统一形式化为如下公式，下式称为 Hammersley-Clifford 定理。

$$P(Y) = \frac{1}{Z} \prod_C \psi_C(Y_C) \quad (4.6.1)$$

$$Z = \sum_Y \prod_C \psi_C(Y_C) \quad (4.6.2)$$

其中， C 是 G 上的最大团集合， $\psi_C(Y_C)$ 是 C 上定义的严格正函数，这就是我们常说的势函数 (Potential Function)。因子分解是在无向图上所有的最大团上进行的。

4.6.3 线性链条件随机场

条件随机场定义：设 $X=(X_1, X_2, \dots, X_n)$ 和 $Y=(Y_1, Y_2, \dots, Y_m)$ 都是联合随机变量，若随机变量 Y 构成一个无向图 $G=(V, E)$ 表示的马尔科夫随机场 (MRF)，则其条件概率分布 $P(Y|X)$ 称为条件随机场 (Conditional Random Field, CRF)，即 $P(Y_v | X, Y_w, w \neq v) = P(Y_v | X, Y_w, w \approx v)$ 。其中， $w \approx v$ 表示与节点 v 相连的所有节点 w 。

目前自然语言处理中，最常用的是线性链条件随机场 (Linear Chain Conditional Random Field)。此时，条件概率 $P(Y|X)$ 中， Y 表示标记序列 (或状态序列)， X 是被标注的观测序列。

如图 4.16 所示，线性链条件随机场定义：设 $X=(X_1, X_2, \dots, X_n)$ 和 $Y=(Y_1, Y_2, \dots, Y_n)$ 均为线性链表示的随机变量序列，若在给定的随机变量序列 X 的条件下，随机变量序列 Y 的条件概率分布 $P(Y|X)$ 构成条件随机场，即满足马尔科夫性：

$$P(Y_i | X, Y_1, Y_2, \dots, Y_n) = P(Y_i | X, Y_{i-1}, Y_{i+1})$$

则称 $P(Y|X)$ 为线性链的条件随机场。在自然语言的标注问题中， Y 表示标记序列 (或状态序列)， X 表示输入序列或观测序列。

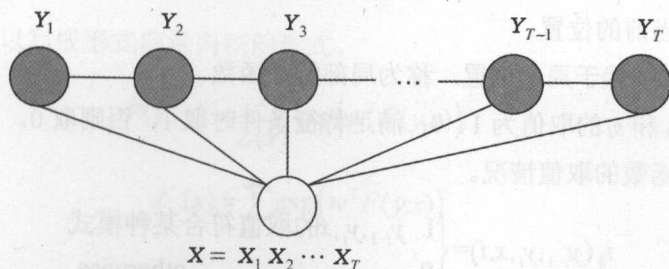


图 4.16 无向图的条件随机场模型

线性链条件随机场的参数化形式如下。

设 $P(Y|X)$ 为线性链条件随机场, 则在随机变量 X 取值为 x 的条件下, 随机变量 Y 取值为 y 的条件概率有如下形式。

$$P(y|x) = \frac{1}{Z(x)} \exp \left(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i) \right) \quad (4.6.3)$$

其中, 归一化因子 $Z(x)$ 为

$$Z(x) = \sum_y \exp \left(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i) \right) \quad (4.6.4)$$

需要注意的是, 这里的式 (4.6.3) 与式 (4.6.1) 是等价的, 式 (4.6.4) 与式 (4.6.2) 也等价。函数之所以做出这样的变形, 是因为任何一个形如:

$$Y = a \cdot X_1 \cdot X_2 \cdots X_n$$

的正函数 (其中, $\forall a, X_i, Y > 0, X_i \in X_n$), 都可以写成形如:

$$Y = \exp(\log(a) + \log(X_1) + \log(X_2) + \cdots + \log(X_n))$$

的形式, 函数没有发生任何变化, 但是参数 X_n 间的相乘关系变为相加关系。这是为了后面推导和寻优方便。

上式中, λ_k 和 μ_l 是对应的权值。 t_k 和 s_l 是特征函数, 这里详细讲解一下特征函数, i 表示 y 的所有可能取值, k 和 l 表示特征函数的个数。其中, $t_k(y_{i-1}, y_i, x, i)$ 中的 y_{i-1}, y_i, x 三个节点构成线性链上的最大团, $s_l(y_i, x, i)$ 表示不同 x 标注为 y_i 的概率。

综上所述, 总结出条件随机场的参数如下。

- t_k 是定义在 (y_{i-1}, y_i, x) 边上的特征函数, 称为转移特征, 依赖于当前和前一个位置。当然也可以是后面的位置。 s_l 是定义在 (y_i, x) 上的特征函数, 称为状态特征, 仅依赖于当前的位置。
- t_k 和 s_l 都依赖于局部位置, 称为局部特征函数。
- 通常, t_k 和 s_l 的取值为 1、0; 满足特征条件时取 1, 否则取 0。下面以 t_k 为例说明特征函数的取值情况。

$$t_k(y_{i-1}, y_i, x, i) = \begin{cases} 1 & y_{i-1}, y_i, x \text{ 的取值符合某种模式} \\ 0 & \text{otherwise} \end{cases}$$

s_l 的取值情况与此相同。

- CRF 完全由特征函数 t_k 、 s_l ，以及与之相对应的权值 λ_k 和 μ_l 确定。
- 线性链条件随机场的函数使用了对数变换，因此称这种变换后的模型为对数线性模型。

上式有些复杂，不便于推导，为了表示方便，将其逐步简化。

(1) t_k 和 s_l 都依赖于局部位置，为方便起见，可以将转移特征和状态特征及其权值用统一的符号表示。

设有 K_1 个转移特征，有 K_2 个状态特征， $k=K_1+K_2$ ，则：

$$f_k(y_{i-1}, y_i, x, i) = \begin{cases} t_k(y_{i-1}, y_i, x, i) & k=1, 2, \dots, K_1 \\ s_l(y_i, x, i) & k=K_1+l, l=1, 2, \dots, K_2 \end{cases}$$

对上式进一步简化：

$$f_k(y, x) = \sum_{i=1}^n f_k(y_{i-1}, y_i, x, i), \text{ 其中, } k=1, 2, \dots, K. \quad (4.6.5)$$

(2) 用 w 来表示统一的权重 λ_k 和 μ_l 。

$$w = \{w_1, w_2, \dots, w_K\}$$

(3) 简化后的 CRF 可以表示为如下形式。

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{k=1}^K w_k f_k(y, x)\right) \quad (4.6.6)$$

$$Z(x) = \sum_y \exp\left(\sum_{k=1}^K w_k f_k(y, x)\right) \quad (4.6.7)$$

(4) 将 $f(y, x)$ 写成向量的形式。

$$F(y, x) = \{f_1(y, x), f_2(y, x), \dots, f_K(y, x)\}$$

(5) CRF 可以写成形式向量内积的形式。

$$P(y|x) = \frac{1}{Z(x)} \exp(w^T F(y, x)) \quad (4.6.8)$$

$$Z_w(x) = \sum_y \exp(w^T F(y, x)) \quad (4.6.9)$$

(6) CRF 的矩阵形式如下。

引入起始、终止节点标记。

① $y_0=\text{start}$, $y_{n+1}=\text{stop}$ 。

start 和 stop 为标记空间的某两个值。

② 定义 m 阶矩阵 (m 是标记 y_i 取值的个数)。

$$M_i(x)=[M_i(y_{i-1},y_i|x)]$$

$$M_i(y_{i-1},y_i|x)=\exp(W_i(y_{i-1},y_i|x))$$

$$W_i(y_{i-1},y_i|x)=\sum_{k=1}^K w_k f_k(y_{i-1},y_i,x,i)$$

$$\text{此时, 条件概率 } p(y|x) \text{ 就变成: } P_w(y|x)=\frac{1}{Z_w(x)} \prod_{i=1}^{n+1} M_i(y_{i-1},y_i|x)$$

$$\text{其中, } Z(x) \text{ 是归一化因子: } Z_w(x)=(M_1(x)M_2(x)\cdots M_{n+1}(x))_{\text{start,stop}}$$

$y_0=\text{start}$, $y_{n+1}=\text{stop}$ 表示开始和终止状态, 是以 start 为起点、 stop 为终点通过状态 y_1, y_2, \dots, y_n 为所有路径的非规范化概率之和, 也是矩阵连乘 $M+1$ 次后得到的新矩阵 (start, end) 位置的元素值。

4.6.4 CRF 的概率计算

给定条件随机场 $P(Y|X)$, 输入序列 x 和输出序列 y , 使用前向、后向算法都到计算的 $P(Y_i=y_i|x)$ 和 $P(Y_{i-1}=y_{i-1}, Y_i=y_i|x)$ 。

(1) 前向向量。

定义 $\alpha_i(y_i|x)$: 输入序列位置 i 的标记是 y_i , 并且到位置 i 之前部分标记序列的非归一化概率。如果 y_i 可取的值有 m 个, $\alpha_i(y_i|x)$ 是 m 维列向量, 则

$$\alpha_0(y|x)=\begin{cases} 1 & y=\text{start} \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha_i^T(y_i|x)=\alpha_{i-1}^T(y_{i-1}|x)M_i(y_{i-1},y_i|x) \quad i=1,2,\dots,n+1$$

$$\alpha_i^T(x)=\alpha_{i-1}^T(x)M_i(x)$$

(2) 后向向量。

定义 $\beta_i(y_i|x)$: 输入序列位置 i 的标记是 y_i , 并且到位置 $i+1$ 到 n 之后部分标记序列的非归一化概率。如果 y_i 可取的值有 m 个, $\beta_i(y_i|x)$ 是 m 维列向量, 则

$$\beta_{n+1}(y_{n+1} | x) = \begin{cases} 1 & y_{n+1} = \text{stop} \\ 0 & y_{n+1} \neq \text{stop} \end{cases}$$

$$\beta_i(y_i | x) = M_i(y_i, y_{i+1} | x) \beta_{i+1}(y_{i+1} | x) \quad i=1, 2, \dots, n+1$$

$$\beta_i(x) = M_{i+1}(x) \beta_{i+1}(x)$$

(3) 计算归一化因子 $Z(x)$ 。

$$Z(x) = \alpha_n^T(x) \cdot 1 = 1^T \cdot \beta_1(x)$$

(4) 计算 $P(Y_i=y_i|x)$ 和 $P(Y_{i-1}=y_{i-1}, Y_i=y_i|x)$ 。

$$P(Y_i=y_i|x) = \frac{\alpha_i^T(y_i|x) \beta_i(y_i|x)}{Z(x)} \quad (4.6.10)$$

$$P(Y_{i-1}=y_{i-1}, Y_i=y_i|x) = \frac{\alpha_{i-1}^T(y_{i-1}|x) M_i(y_{i-1}, y_i|x) \beta_i(y_i|x)}{Z(x)} \quad (4.6.11)$$

4.6.5 CRF 的参数学习

根据训练数据集 (x, y) ，可以求出经验概率分布 $P(x, y)$ ，然后使用极大似然估计给出目标函数。这里未知数是 w 。

$$L(w) = \log \prod_{x, y} p_w(y | x)^{\tilde{p}(x, y)} = \sum_{x, y} \tilde{p}(x, y) \log p_w(y | x) \quad (4.6.12)$$

$$= \sum_{x, y} \left[\tilde{p}(x, y) \left(\sum_{k=1}^K w_k f_k(x, y) \right) - \tilde{p}(x, y) \log Z_w(x) \right]$$

$$= \left(\sum_{j=1}^N \sum_{k=1}^K w_k f_k(x_j, y_j) \right) - \sum_{j=1}^N \log Z_w(x_j)$$

通过似然函数 w 变量的下界，给定学习速率 δ ，通过迭代法寻求最优的参数 w 。

$$w = (w_1, w_2, \dots, w_K)^T$$

$$\delta = (\delta_1, \delta_2, \dots, \delta_K)^T$$

$$w + \delta = (w_1 + \delta_1, w_2 + \delta_2, \dots, w_K + \delta_K)^T$$

经过计算，转移特征 t_k 的更新方程为

$$E_P[t_k] = \sum_{x, y} \tilde{p}(x) p(y|x) \left(\sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k T(x, y)) \right) \quad k=1, 2, \dots, K_1 \quad (4.6.13)$$

状态特征 s_l 的更新方程为:

$$E_p[s_l] = \sum_{x,y} \tilde{p}(x) p(y|x) \left(\sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} T(x, y)) \right) \quad l=1, 2, \dots, K_2 \quad (4.6.14)$$

其中, $T(x, y)$ 为在数据 (x, y) 中出现的所有特征数之和。

$$T(x, y) = \sum_{k=1}^K \sum_{i=1}^{n+1} f_k(y_{i-1}, y_i, x, i)$$

当前转移特征 t_k 和状态特征 s_l 给定的条件下, 求关于未知向量 δ 。取所有 $T(x, y)$ 的最大值 S , 并用 S 代替其他 $T(x, y)$, 则上式可直接给出如下解析式:

$$\delta = \frac{1}{S} \log \frac{E_p[t_k]}{E_p[t_k]}$$

其中,

$$E_p[t_k] = \sum_{x,y} \tilde{p}(x) \left(\sum_{i=1}^{n+1} \sum_{y_{i-1}, y_i} t_k(y_{i-1}, y_i, x, i) \frac{\alpha_{i-1}^T(y_{i-1}|x) M_i(y_{i-1}, y_i|x) \beta_i(y_i|x)}{Z(x)} \right)$$

为了追求更高的精度, 常常使用基于拟牛顿法的 L-BFGS 等方法计算得到 δ 。第 5 章讲解的 CRF++ 框架就是使用 L-BFGS 方法训练出模型的参数。

4.6.6 CRF 预测标签

给定条件随机场 $P(Y|X)$ 和输入序列 x , 求条件概率最大的输出序列 (标记序列) y^* , 即对观测序列进行标注。这一过程仍旧使用 HMMs 的 Viterbi 算法。

优化目标函数:

$$\begin{aligned} y^* &= \operatorname{argmax}_y P_w(y|x) \\ &= \operatorname{argmax}_y \frac{\exp(wF(y, x))}{Z_w(x)} \\ &= \operatorname{argmax}_y \exp(wF(y, x)) \\ &= \operatorname{argmax}_y wF(y, x) \end{aligned}$$

推导之后可知, CRF 的预测问题就是非规范化概率最大的最优路径问题。

其中,

$$w=\{w_1, w_2, \dots, w_K\}$$

$$F(y, x)=\{f_1(y, x), f_2(y, x), \dots, f_k(y, x)\}$$

$$f_k(y, x)=\sum_{i=1}^n f_k(y_{i-1}, y_i, x, i)$$

根据 Viterbi 算法, 位置 i 的各个标记 $l=1, 2, \dots, m$ 的非规范化概率的最大值。

(1) 起始位置的概率计算: $\delta_1=w \cdot F_1(y_0=\text{start}, y_1=l, x), l=1, 2, \dots, m$ 。

(2) 中间位置的概率计算: $\delta_l(l)=\max_{1 \leq j \leq m} \{\delta_{l-1}(j)+w \cdot F_l(y_{l-1}=j, y_l=l, x)\}, l=1, 2, \dots, m$ 。

(3) 最优路径: $y_n^*=\operatorname{argmax}_{1 \leq j \leq m} \delta_n(j)$ 。

4.7 结语

关于 NLP 的算法理论, 全书共分为两章讲解, 本章为第一章: 概率图模型。本章重点讲解了如下 4 个算法: 朴素贝叶斯模型、隐马尔科夫模型、最大熵模型和条件随机场模型。

这 4 个模型是 NLP 中常用的算法系列之一。由于 4 个模型的数学理论联系密切, 且一脉相承, 因此, 把它们放到一起, 有助于读者全面地掌握其整体的数学思想。

朴素贝叶斯模型、隐马尔科夫模型和最大熵模型, 对于这三个模型本书不仅列出算法公式、推导过程, 还给出了实际的案例代码。由于篇幅有限, 这里没有给出 CRF 模型的案例, 第 5 章将会详细讲解 CRF 在 NLP 中的应用, 并介绍一些高级应用的示例。

第 5 章

词性、语块与命名实体识别

自然语言处理的很多问题，要求解的输出标签不是相互独立的，而是时间或结构上相互依存的结构化标签。这种结构包括序列、树状或更普通意义上的图结构。而对于中文分词、词性标注、组块标注、浅层语法分析等任务，标记和切分观察序列都是序列结构的。解决此类方法最常用的模型也是概率图模型中的序列算法。因此，这一系列问题通常称作标记序列学习或序列学习。使用序列模型算法解决的 NLP 标注问题也统一称为序列标注任务。

除算法角度之外，序列标注任务的提出也有其应用层面的价值。在 NLP 分析技术中，序列标注主要用于句子的浅层分析（相对于完全句法分析而言），即对句子的局部范围进行分析处理。由于以概率图模型为主的算法能够提供高精度的分析结果，这里技术已经基本成熟，其标志就是其已经被成功地用于文本检索、文本分类、信息抽取等应用之中，并对这些应用产生了实质性的帮助。

本章所讲的三类问题都是以概率图模型为核心算法的浅层分析（序列标注任务）。从自然语言处理的流程来看，任务的顺序应该为，首先进行命名实体识别，它常作为中文分词的子模块，然后进行词性标注，最后才是语义组块标注。但从算法应用策略的复杂性而言，我们在顺序上做了调整。最先介绍的是词性标注，之后是语义组块标注，最后是命名实体识别。因为，词性标注仅仅使用了最大熵算法即可得到比较高精度的结果；组块标注最常用的算法是条件随机场算法，在特征上加入了词性之后，结果也不错；命名实体识别最复杂，在算法策略上，使用了含有外部词典的条件随机场算法（标准条件

随机场算法的一个变种)及多层隐马尔科夫模型(张华平)两种算法。在三种任务中,命名实体识别是最复杂的序列标注任务,所以放到最后来讲。

需要说明的是,从本章开始,包括后面的一些章节,很多实例用到了一些收费资源,如美国宾州大学开发的汉语句法树库(Chinese Tree Bank, CTB, 项目编号为LDC2013T21)及汉语命题库(Chinese Proposition Bank, CPB, 项目编号为LDC2013T13)。这两种语料都为美国宾州大学开发,一个是中文短语结构树库,另一个是谓词论元库。注意,CPB不能直接使用。读者必须首先购买CTB的语料资源,然后通过程序将CPB的参数映射到CTB上,才能得到完整的命题资源。有关CPB转换、使用等更多细节问题,在后面还有介绍,相关代码也将发布出来。宾州大学定期更新两种语料资源,目前CTB的最高版本为8.0,CPB为3.0。如果读者没有这两种语料,那么可以通过NLTK下载该语料(CTB)的英文样例版,样例版大约有1000句(仅为了学习应该足够了)。

本节使用了CPB标注方法为短语结构树(可通过程序转为依存句法树)。该语料为收费语料,价格不贵(我们不是做广告,因此就不详细介绍了)。

5.1 汉语词性标注

第1章和第3章都介绍过汉语分词模块。第3章还通过《北大规范》词性表引入了词性的一些基本标签。从本节开始对汉语词性标注进行详细介绍。

5.1.1 汉语的词性

什么是词性(Part-of-Speech, POS)呢?词性,也称为词类,是词汇的语法属性,是连接词汇到句法的桥梁。由于汉语缺乏形态变化,一个词的词性与它在句子中的成分密切相关。

游泳是一种很好的健身运动(名词,表示这一运动)。

下午,我去体育馆游泳(动词,表示这一动作)。

同一个词,因为在句子中扮演的角色不同,其词性也不相同。我们都知道,在传统的语法学中,汉语句子分为主语、谓语、宾语、定语、状语、补语6个成分。单从词性的角度来看,主语和宾语都是名词性结构,多是名词、代词;谓语则多是动词、形容词;

定语修饰、限制名词性结构，多是名词、代词、数量词、形容词；状语修饰、限制动词或形容词，多是形容词、副词、数量词；补语是动词或形容词后面的连带成分，也多是形容词、副词、数量词。因此，从句法的角度来看，一个词的词性是由它在句子中的成分所决定的。这才是划分词类的依据。

一个词的词类（词性）是指，在一个语言中，众多具有相同句法功能、能在同样的组合位置中出现的词，聚合在一起形成的范畴。词类是最普遍的语法的聚合。汉语中的词类划分具有层次性。大的分类可以分成实词和虚词，实词又包括体词、谓词等，体词中又可以分出名词和代词等。现代汉语的词可以分为两类 12 种词性（一说是 14 种）。一类是实词：名词、动词、形容词、数词、量词和代词。另一类是虚词：副词、介词、连词、助词、叹词和拟声词。这是传统语法学对汉语词汇的类别标准。计算语言学界中一些更复杂的分类标准也是以此作为基准的。

另一方面，在后面章节的语义角色标注中，词性对判断句子的语义角色也起着至关重要的作用。因此，无论是对于句法分析还是对于语义分析，对词性标注都是自然语言处理的一项重要基础工作。

关于词性标注歧义的问题，英文中根据对 Brown 语料库进行统计，按歧义程度排列的词型数目 DeRose（1988）给出了表 5.1 所示的 Brown 语料库词性标记歧义表。

表 5.1 Brown语料库词性标记歧义表

无兼类只有1个标记	35 340
有兼类有2~7个标记	4 100
2个标记	3 764
3个标记	264
4个标记	61
5个标记	12
6个标记	2
7个标记	1

由表 5.1 可知，英语中的大多数单词都是没有歧义的，也就是这些单词只有一个单独的标记。但是，汉语与印欧语言不同，汉语缺乏丰富的形态变化，无法从词的形态变化来判别汉语词汇的词性。而且，汉语中常用词的兼类现象非常严重。根据对北京大学计算语言学研究所在网上公布的 200 万个汉字语料进行统计，其中兼类词占 11%，这和英语差不多，但是这些兼类词的词频却占总词频的 47%（张虎等人，2004）。因此，虽然兼类词占汉语词汇很小的一部分，但是由于兼类使用的程度高，兼类现象复杂，范围

面广，因此，汉语的词性标注（POS Tagging）的主要任务与汉语分词差别不大，主要仍是消歧问题。

5.1.2 宾州树库的词性标注规范

虽然大家都承认 12 种词类的基准分类，但在计算语言学界，词类划分的很多细节却从未统一过。中科院 ICTCLAS 和前文的《北大规范》的词类总数都为 39 个。北大计算语言学研究所开发的语料库加工规范中，基本词类代码有 26 个，后来又增加了 74 个扩充代码，标记集中共计 104 个。而各个大型网站的词性标注都各有各的标准，最少的有 17 个，最多的有 173 个。单从数字上看，虽然标准复杂，但仔细观察后发现大同小异：有的是《北大规范》的精简版，有的则以《北大规范》为基础，增加了多个类别层次。

那么，我们使用哪种分类标准呢？从词性对句子结构的标识作用来看，本书使用的词性标注规范为宾州树库（Penn Treebank）中的《汉语词性标注规范》，以下简称《宾州规范》。表 5.2 所示为宾州树库的词性标注规范。

表 5.2 宾州树库的词性标注规范

词性 标记	英文名称	中文名称	例 子
AD	adverbs	副词	“还”
AS	Aspect marker	体标记	了, 着, 过, “的 (我是去年来的)”
BA	in ba-const	把/将	把, 将
CC	Coordinating conjunction	并列连词	“和” “与” “或” “或者”
CD	Cardinal numbers	数字/基数词	“一百”
CS	Subordinating conj	从属连词	若, 如果, 如
DEC	for relative-clause etc	标句词, 关系从句“的”	我买 “的” 书
DEG	Associative	所有格/联结作用“的”	我 “的” 书
DER	in V-de construction, and V-de-R	V得, 表示结果补语的 “得”	跑 “得” 气喘吁吁
DEV	before VP	表示方式状语的 “地”	高兴/VA 地/DEV 说/VV
DT	Determiner	限定词	这
ETC	Tag for words in coordination phrase	“等”, “等等”	科技文教 等 /ETC 领域 “等”, “等等”
FW	Foreign words	外语词	ISO
IJ	interjection	感叹词	啊

续表

词性 标记	英文名称	中文名称	例 子
JJ	Noun-modifier other than nouns	其他名词修饰语	共同/JJ 的/DEG目的/NN, 她/PN 是/VC 女/JJ 的/DEG
LB	in long bei-construction	长“被”	“被”他打了
LC	Localizer	方位词	桌子“上”
M	Measure word (including classifiers)	量词	一“间”房子
MSP	Some particles	其他结构助词	他/PN 所/MSP 需要/VV 的/DEC。 所, 而, 以, 而
NN	Common nouns	其他名词, 普通名词	桌子
NR	Proper nouns	专有名词	北京
NT	Temporal nouns	时间名词	一月, 汉朝
OD	Ordinal numbers	序数词	“第一”
ON	Onomatopoeia	拟声词	“哗啦啦”
P	Prepositions (excluding把and被)	介词	“在”
PN	pronouns	代词	“你”, “我”, “他”
PU	Punctuations	标点	, 。
SB	in long bei-construction	短被	他被/SB 训了/AS一顿/M
SP	Sentence-final particle	句末助词	他好吧[SP] 了, 呢, 吧, 啊, 呀, 吗
VA	Predicative adjective	谓形容词	花很 红/VA 红彤彤 雪白 丰富
VC	Copula	系动词	“是”“为”“非”
VE	as the main verb	“有”作为主要动词	“有”, “无”
VV	Other verbs	其他动词, 普通动词	走, 可能, 喜欢

宾州规范的词性标注标记集有如下 33 种。动词和形容词 (4): VA、VC、VE、VV。名词 (3): NR、NT、NN、方位词 (1): LC。代词 (1): PN。限定词和数词 (3): DT、CD、OD。度量词 (1): M。副词 (1): AD。介词 (1): P。连词 (2): CC、CS。助词 (8): DEC、DEG、DER、DEV、SP、AS、ETC、MSP。其他 (8): IJ、ON、PU、JJ、FW、LB、SB、BA。

《宾州规范》在实词上与《北大规范》大同小异,但在虚词上差异较大。下面给出《宾州规范》词性标注的详细说明(读者可以以此与《北大规范》的虚词进行比较)。

1. 虚词: DEC、DEG、DER、DEV、AS、SP、ETC、MSP

1) “的”作为补语标记/名词化标记: DEC (的, 之)

例如, 吃的 DEC。

模式: S/VP DEC {NP}。

注: “的”还有其他标记。

□ DEC: 他 的/DEG 车。

□ SP: 他 是/VC 一定 要 来 的/SP。

□ AS: 他 是/VC 在 这里 下 的/AS 车。

2) “的”作为关联标记或所有格标记: DEG

模式: NP/PP/JJ/DT DEG {NP}。

3) 补语短语 得: DER

在 V-得-R 和 V-得结构中, “得”标记为 DER。

注: 有些以“得”结尾的搭配不是 V-得结构, 如记得、获得是动词。

4) 方式“地”: DEV

当“地”出现在“XP 地 VP”中时, XP 修饰 VP。在一些古典文学中, “的”也用于这种情景, 此时“的”也标注为 DEV。

5) 动态助词: AS

动态助词仅包括“着、了、过、的”。

6) 句末助词: SP

SP 经常出现在句末。例如, 他好吧[SP]?

有时, 句末助词用于表停顿。例如, 他 吧[SP], 人 很 好。

例如, 了、呢、吧、啊、呀、吗。

7) ETC

ETC 用于标注“等”、“等等”。

8) 其他助词: MSP

“所、以、来、而”, 当它们出现在 VP 前时, 标注为 MSP。

所：他 所[MSP] 需要 的/DEC。

以或来：用……以/MSP（或来）维持。

而：为……而[MSP]奋斗。

2. “被”字结构

1) 长“被”结构：LB

仅包括“被、叫、给、为（口语中）”，当它们出现在被字结构 NP0+LB+NP1+VP 中。

例如，他被/LB 我训了/AS 一顿/M。

注：当“叫”作为兼语动词时，“叫”标注为 VV。

例如，他叫/VV 你去。

2) 短“被”结构：SB（仅包括口语中的“被、给”）

NP0+SB+VP，他 被/SB 训了/AS 一顿/M。

注：“给”有其他标记：LB、VV 和 P。

例如，你 给/P 他 写 封/M 信。

3. “把”字结构：BA

仅包括“把、将”，当它们出现在把字结构（NP0+BA+NP1+VP）中。

例如，他 把/BA 你 骗 了/AS。

注：“将”有其他标记，AD 和 VV。例如，他 将/VV 了/AS 我 的/DEG 军。

4. 其他名词修饰语：JJ

包括如下三种类型。

(1) 区别词。只修饰模式 JJ+的+{N} 或 JJ+N 中的名词，且一定要有“的”，它们不能被程度副词修饰。

例如，共同/JJ 的/DEG 目标/NN，她是[VC]女/JJ 的/DEG。

(2) 带有连字符的复合词。

通常为双音节词 JJ+N。例如，留美/JJ 学者/NN。

(3) 形容词。新/JJ 消息/NN。

模式：JJ+N。

注：当“的/DEC”在形容词和名词中间时，形容词标记为 VA。

5. 外来词：FW

FW 仅用于：当词性标注标记在上下文中不是很清楚时。外来词不包括外来词的翻译，不包括混合中文的词（如卡拉 OK/NN、A 型/NN），不包括词义和词性在文中都是清楚的词。此外，《宾州规范》还给出了与《北大规范》的词性转换对照表（见表 5.3）

表 5.3 《宾州规范》与《北大规范》的词性转换对照表

中文词性名称	英文标签	宾州规范	北大规范
总标签数	<i>total tags</i>	33	26
名词	<i>nouns</i>	3	3
时间名词	temporal noun	NT	t
动名词	verbal noun	NN	V[+nom]
专有名词	proper noun	NR	n
其他名词	other noun	NN	n、s
方位词	<i>localizer</i>	LC	f
代词	<i>pronouns</i>	PN	m
动词	<i>verbs</i>	4	3
系动词	shi4	VC	V
“有”动词	you3	VE、VV	v、a、z
其他动词	other verbs	W、VA	V
副词	<i>Adverb</i>	AD	d
介词	<i>prepositions</i>	P	p
限定词及其相关	<i>DP-related</i>	4	2
限定词	determiner	DT	r
数词	number	CD、OD	m
量词	measure word	M	q
连词	<i>conjunctions</i>	2	4
并列连词	coord, conj	CC	c
从属连词	subord. conj	CS	c
助词	<i>particles</i>	8	2
体标记	aspect marker	AS	u
助词	的	DEC、DEG、AS、SP	u

续表

中文词性名称	英文标签	宾州规范	北大规范
助词	地	DEV	u
助词	得	DER	u
句末助词	sent-final part.	SP	y
助词	等	ETC	??
其他助词	other particles	MSP	u
其他词类	<i>others</i>	8	4
叹词	interjection	IJ	e
拟声词	sound word	ON	o
标点符号	punctuation	PU	w
名词修饰成分	noun-modifier	JJ	b
外来词	foreign words	FW	??
“被”字结构	被	LB、SB	p
“把”字结构	把	BA	p

5.1.3 stanfordNLP 标注词性

中文自动词性标注的算法很多，基本都属于概率图模型的范畴，这部分的算法思想前面已经讲过。本节所使用的是基于最大熵的词性标注算法。在诸多系统中，最著名的是 stanfordNLP 的词性标注模块。下面简要介绍 stanfordNLP 词性标注模块的自动标注和模型训练过程。

第 1 章简单介绍过一些开源的词性标注系统。其中，使用 Python 整合了 Stanford Postagger 模块进行词性标注。本节给出 stanford-corenlp 项目的 Java 构建方法。

- (1) 软件包下载：有关 Stanford 软件包的安装在第 1 章中讲过一部分，本节略。
- (2) 项目安装。

Stanford 的所有项目均构建在 Java 的 Eclipse IDE 下。读者可以创建一个 Java 项目，并确定项目名称，名称统一为 stanfordNLP。工作空间和项目的字符集均使用 UTF-8（Windows/Linux）。使用的包为第 1 章讲过的 stanford-corenlp，下载文件名为 stanford-corenlp-full-2015-12-09.zip，汉语模型包为 stanford-chinese-corenlp-2015-12-08-models.jar。下面给出所有的依赖包和语言模型包目录（见图 5.1）。

项目依赖的lib目录	项目语言模型目录
<div><div>stanfordNLP</div><div><div>> src</div><div>> JRE System Library [JavaSE-1.8]</div><div>> ejml-0.23.jar</div><div>> javax.json-api-1.0-sources.jar</div><div>> javax.json.jar</div><div>> joda-time-2.9-sources.jar</div><div>> joda-time.jar</div><div>> jollyday-0.4.7-sources.jar</div><div>> jollyday.jar</div><div>> protobuf.jar</div><div>> slf4j-api.jar</div><div>> slf4j-simple.jar</div><div>> xom-1.2.10-src.jar</div><div>> xom.jar</div><div>> com.sun.net.httpserver.jar</div></div></div>	<div><div>models</div><div><div>> dcoref</div><div>> hcoref</div><div>> lexparser</div><div>> ner</div><div>> parser</div><div>> pos-tagger<div><div>chinese-distsim</div><div>chinese-distsim.tagger</div><div>chinese-distsim.tagger.props</div></div></div><div>> segmenter</div><div>> srparser</div></div></div> <div>该目录的文件解压自 stanford-chinese-corenlp-2016-01-19-models.jar 的汉语模型文件。</div>
源代码目录	
<div><div>stanfordNLP</div><div><div>src</div><div><div>> edu.stanford.nlp.classify</div><div>> edu.stanford.nlp.dcoref</div><div>> edu.stanford.nlp.dcoref.sievepasses</div><div>> edu.stanford.nlp.fsm</div><div>> edu.stanford.nlp.graph</div><div>> edu.stanford.nlp.hcoref</div><div>> edu.stanford.nlp.hcoref.data</div><div>> edu.stanford.nlp.hcoref.docreader</div><div>> edu.stanford.nlp.hcoref.md</div><div>> edu.stanford.nlp.hcoref.rf</div><div>> edu.stanford.nlp.hcoref.sieve</div><div>> edu.stanford.nlp.ie</div><div>> edu.stanford.nlp.ie.crf</div><div>> edu.stanford.nlp.ie.machinereading</div><div>> edu.stanford.nlp.ie.machinereading.cor</div></div></div></div>	

图 5.1 stanford-corenlp 依赖包和语言模型包

图 5.1 中下图为部分源代码目录。使用此框架的一个重要原因是其运行的词性标注结果遵循宾州树库的词性标注规范。

我们编写了一个词性标注的测试例子程序，其他的测试例子版本参见 <http://new.galalaly.me/2011/05/tagging-text-with-stanford-pos-tagger-in-java-applications/>。

```
package edu.stanford.testdemo;

import java.io.BufferedReader;
import java.io.StringReader;
import java.util.ArrayList;
```

```

import java.util.List;

import edu.stanford.nlp.ling.HasWord;
import edu.stanford.nlp.tagger.maxent.MaxentTagger;

public class PostagDemo {

    public static void main(String[] args) {
        //标注
        String model = "models/pos-tagger/chinese-distsim/chinese-distsim.tagger";
        String content = "你们是祖国美丽盛开的花朵";
        MaxentTagger tagger = new MaxentTagger(model);
        List<List<HasWord>> sentences = MaxentTagger.tokenizeText(new BufferedReader
(new StringReader(content)));
        for (List<HasWord> sentence : sentences) {
            List<edu.stanford.nlp.ling.TaggedWord> tSentence = tagger.tagSentence
(sentence);
            System.out.println(tSentence);
        }
    }
}

```

输出结果如下。

```

Reading POS tagger model from models/pos-tagger/chinese-distsim/chinese-distsim.
tagger ... done [0.6 sec].
[你们/PN, 是/VC, 祖国/NN, 美丽/VA, 盛开/VV, 的/DEC, 花朵/NN]

```

在 Stanford 中文词性标注模型中，可以使用两个类词性标注模型，分别为 `chinese-distsim.tagger` 和 `chinese-nodistsim.tagger`。

❑ `chinese-distsim.tagger`。

语料：基于来自中国内地和中国香港的 CTB 7.0 语料上训练得到，并做了相似性聚类。

标注集：中文 LDCPOS 树库标记集。

性能：对中文和中国香港文本的识别精度为 93.99%（未知词为 84.60%）。

❑ `chinese-nodistsim.tagger`。

语料：同样基于来自中国内地和中国香港的 CTB 7.0 语料上训练得到，但未做相似性聚类。

标注集：中文 LDCPOS 树库标记集。

性能：对中文和中国香港文本的识别精度为 93.46%（未知词为 79.40%）。

5.1.4 训练模型文件

在实际应用中，用户一般都根据自己的需求，自定义自己的模型。下面简要介绍模型的训练方法。

在模型文件的 `chinese-distsim` 目录下，有一个名为 `chinese-distsim.tagger` 的文件。它是 `stanford` 词性标注的模型文件，与它并列的还有一个训练时使用的配置文件 `chinese-distsim.tagger.props`。我们对该文件做一些简单的修改即可完成训练过程。但对刚刚接触这个库的读者而言，很多参数可能都不太熟悉。为了使读者能够加深了解每个参数的意义，我们专门编写了一个程序，用于生成样本的 `props` 配置文件。该文件对于每个配置选项都提供了完整的注释，便于读者进一步学习。

首先，回顾一下第 1 章中使用过的两个 `python` 类：`StanfordCoreNLP` 类和其子类 `StanfordPOSTagger` 类。`StanfordPOSTagger` 类专门用于处理词性标注。下面在这个类中添加如下两个方法。

(1) `buildprop` 方法：用来创建属性文件的命令行指令。

```
def __buildprop(self): #创建属性文件
    self.propline = 'java -mxlg -cp "'+self.jarpath+'" '+self.classfier+'
-genprops'
```

(2) `genpropfile` 方法：用来输出创建好的属性文件。

```
def genpropfile(self,propath): # 获取属性文件
    self.__buildprop()
    propfile = os.popen(self.propline,'r').read()
    self.savefile(propath,propfile)
    print "save properties to ",propath
```

然后，输出创建的结果，内容如下。

```
# -*- coding: utf-8 -*-
import sys
import os
from stanford import StanfordPOSTagger

reload(sys) # 设置 UTF-8 输出环境
```



```

sys.setdefaultencoding('utf-8')

# print os.environ
root = 'X:/nltk_data/stanford-corenlp/'
modelpath = root+"models/pos-tagger/chinese-distsim/chinese-distsim.tagger"
st = StanfordPOSTagger(root,modelpath)
proppath = "my.tagger.props"
st.genpropfile(proppath)

```

输出结果:

在当前目录（你的 python 执行文件目录）下生成一个空的 my.tagger.props。因为创建的文件比较大，所以只给出这个文件的部分代码，内容如下。

```

...
# Model file name (created at train time; used at tag and test time)
# (you can leave this blank and specify it on the commandline with -model)
# model =

# Path to file to be operated on (trained from, tested against, or tagged)
# Specify -textFile <filename> to tag text in the given file, -trainFile <filename> to
# to train a model using data in the given file, or -testFile <filename> to test
your
# model using data in the given file. Alternatively, you may specify
# -dump <filename> to dump the parameters stored in a model or
# -convertToSingleFile <filename> to save an old, multi-file model (specified as
-model)
# to the new single file format. The new model will be saved in the file filename.
# If you choose to convert an old file, you must specify
# the correct 'arch' parameter used to create the original model.
# trainFile =

# Path to outputFile to write tagged output to.
# If empty, stdout is used.
# outputFile =
...

```

受篇幅的限制，我们不可能给出样本配置文件中的各个选项的含义，读者可以根据这个样本的配置文件自行了解。这里仅给出如下几个必要参数的说明。

- ☐ **model**。训练完成后的需要保存的模型文件路径。
- ☐ **trainFile**。训练集文件名。
- ☐ **tagSeparator**。训练集中词汇与标签的分割符（“_”）。

❑ encoding。默认使用 UTF-8 字符集。

❑ arch。特征模板规则: words(-1,1),unicodeshapes(-1,1),order(2),suffix(4)。

words(-1,1): 前一词, 当前词, 后一词。

unicodeshapes(-1,1): unicode 字符集, 规则同上。

order(2): 从前两个标签来预测当前标签。

suffix(4): 试图从当前词的前 4 个字来预测未知词或稀有词的词性。

❑ search。默认优化算法为 owlqn, 但该算法未发布出来, 可选的算法有 owlqn2 或 qn。

❑ iterations。算法迭代次数, 默认为 100。

下面尝试使用该词性标注模块构建一个自己的 mini 模型。

一个 mini 的训练集文件如下。

在_P 包含_VV 问题_NN 的_DEC 所有_DT 解_VV 的_DEC 解空间_NN 树_NN 中_LC, _PU 按照_P 深度优先_NN 搜索_NN 的_DEC 策略_NN, _PU 从_P 根节点_NN 出发_VV 深度_JJ 探索_NN 解空间_NN 树_NN。_PU

手工编辑 my.tagger.props 文件, 产生完整配置文件信息, 内容如下。

```
model = my.model.tagger
arch = generic(suffix(4),prefix(4),unicodeshapes(-1,1),
unicodeshapeconjunction(-1,1),words(-2,-2),words(2,2)
wordFunction = edu.stanford.nlp.util.UTF8EquivalenceFunction
trainFile = train.txt
closedClassTags =
closedClassTagThreshold = 40
curWordMinFeatureThresh = 1
debug = false
debugPrefix =
tagSeparator = _
encoding = utf-8
iterations = 100
lang = chinese
learnClosedClassTags = false
minFeatureThresh = 3
openClassTags =
rareWordMinFeatureThresh = 3
rareWordThresh = 20
```

```

        search = owlqn2
        sgml = false
        sigmaSquared = 0.0
        regL1 = 0.75
        tagInside =
        tokenize = false
        tokenizerFactory =
        tokenizerOptions =
        verbose = false
        verboseResults = true
        veryCommonWordThresh = 250
        xmlInput = null
        outputFile =
        outputFormat = slashTags
        outputFormatOptions =
        nthreads = 1

```

继续修改 `StanfordPOSTagger` 类。在此类中加入生成训练文件的如下两个方法。

(3) `buildmodel` 方法：用来创建模型文件的命令行指令。

```

def __buildtrain(self, propspath): # 创建模型文件
    self.trainline = 'java -mx4g -cp "'+self.jarpath+'" '+self.classfier + '
    -props "'+propspath+'"'

```

(4) `trainmodel` 方法：用来输出创建好的模型文件。

```

def trainmodel(self, propspath): # 训练模型
    self.__buildtrain(propspath)
    os.system(self.trainline)
    print "save model to model.tagger"

```

然后，输出创建的结果，内容如下。

```

# -*- coding: utf-8 -*-
import sys
import os
from stanford import StanfordPOSTagger

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')

# print os.environ
root = 'E:/nltk_data/stanford-corenlp/'
modelpath = root+"models/pos-tagger/chinese-distsim/chinese-distsim.tagger"

```



```
st = StanfordPOSTagger(root,modelpath)
proppath = "my.tagger.props"
st.trainmodel(propspath)
```

输出结果:

在当前目录（你的 python 执行文件目录）下生成了一个 my.model.tagger 的二进制文件。因为是为了演示训练过程，该文件只有 4KB。

最后，给出训练过程的部分输出日志，内容如下。

```
.....
TaggerExperiments: adding word/tags
Loading tagged words from train.txt
Read 25 words from train.txt [done].
Read 1 sentences, min 25 words, max 25 words.
Featurizing tagged data tokens...

Featurized 26 data tokens [done].
xSize [num Phi templates] = 25; ySize [num classes] = 9
Hashing histories ...

Hashed 25 histories.
Hashing populated histories ...

Hashed populated histories.
TaggerExperiments.getFeaturesNew: initializing fnumArr.
  length of sTemplates keys: 175
getFeaturesNew adding features ...
  total feats: 175, populated: 104
  Max features per x,y pair: 9
  Max non-zero y values for an x: 9
  Number of non-zero feature x,y pairs: 192
  Number of zero feature x,y pairs: 33
end getFeaturesNew.
Samples from train.txt
Number of features: 104
Tag set: [P, VV, NN, DT, JJ, DEC, PU, .$$., LC]
pcond initialized
zlambd initialized
ftildeArr initialized
QNNMinimizer called on double function of 104 variables, using M = 10.
```

```

Iter. 0: neg. log cond. likelihood = 57.12783901074175 [1 calls to valueAt]
An explanation of the output:
Iter          The number of iterations
evals         The number of function evaluations
SCALING       <D> Diagonal scaling was used; <I> Scaled Identity
LINESEARCH   [## M steplength] Minpack linesearch
              1-Function value was too high
              2-Value ok, gradient positive, positive curvature
              3-Value ok, gradient negative, positive curvature
              4-Value ok, gradient negative, negative curvature
              [... B] Backtracking
VALUE        The current function value
TIME         Total elapsed time
|GNORM|       The current norm of the gradient
{RELNORM}     The ratio of the current to initial gradient norms
AVEIMPROVE    The average improvement / current value
EVALSCORE     The last available eval score

Iter ## evals ## <SCALING> [LINESEARCH] VALUE TIME |GNORM| {RELNORM} AVEIMPROVE
EVALSCORE

Iter 1 evals 1 <D> [B 1.000E-1] 5.093E1 0.00s
Iter. 1: neg. log cond. likelihood = 48.191508920927475 [2 calls to valueAt]
[5.490E0] {4.211E-1} 0.000E0 -
Iter 2 evals 2 <D> [B 1.000E-1] 4.949E1 0.00s
Iter. 2: neg. log cond. likelihood = 45.91531010406181 [3 calls to valueAt]
[4.733E0] {3.630E-1} 1.456E-2 -
.....
Iter 21 evals 21 <D> [B 1.000E0] 4.229E1 0.02s
Iter. 21: neg. log cond. likelihood = 26.931394983455895 [30 calls to valueAt]
QNMinimizer terminated due to average improvement: | newest_val - previous_val
| / |newestVal| < TOL
Total time spent in optimization: 0.02s
After optimization neg (penalized) log cond likelihood: 26.93
Non-zero parameters: 25/104 (24.04%)
Checking model correctness; x size 25 , ysize 9
Constraint 0 not satisfied emp 0.0769 exp 0.0535 diff 0.0234 lambda 0
Constraint 1 not satisfied emp 0.0769 exp 0.0825 diff 0.0056 lambda 0
.....
Constraint 102 not satisfied emp 0.0769 exp 0.0475 diff 0.0294 lambda 0.3403
Model is not correct
Saving dictionary of 8 words ...

```

```

Extractors list:
Extractors[Extractor(-1,word),      Extractor(0,word),      Extractor(1,word),
ExtractorFrames$ExtractorContinuousTagConjunction(-2,tag), Extractor(-1,tag), edu.
stanford.nlp.tagger.maxent.ExtractorFrames$ExtractorTwoWords(w0,w-1), edu.stanford.
nlp.tagger.maxent.ExtractorFrames$ExtractorWordTag(w0,t-1),      Extractor(-2,word),
Extractor(2,word)]

rareExtractors[ExtractorWordSuff(len1,w0),      ExtractorWordSuff(len2,w0),
ExtractorWordSuff(len3,w0), ExtractorWordSuff(len4,w0), ExtractorWordPref(len1,w0),
ExtractorWordPref(len2,w0), ExtractorWordPref(len3,w0), ExtractorWordPref(len4,w0),
ExtractorWordShapeClassifier(-1,chr4), ExtractorWordShapeClassifier(0,chr4),
ExtractorWordShapeClassifier(1,chr4),
ExtractorWordShapeConjunction(-1,1,chr4)]

Training POS tagger done [0.2 sec].
save model to mymodel.tagger

```

5.2 语义组块标注

法国的著名语言学家 Steven Abney 最早提出了一个完整的组块(Chunk)描述体系,并给出了组块的定义。他把组块定义为句内的一个非递归的核心成分。这种成分包含核心成分的前置修饰成分,而不包含后置附属结构。同时, Abney 还提出了组块解析的策略,通过引进句法块(Chunk)概念,他将句法分析问题分为如下三个阶段。

- ❑ 块识别: 利用块识别器快速识别出句子中所有的块。
- ❑ 块内结构分析: 对每个块内部的成分赋予合适的句法结构。
- ❑ 块间关系分析: 利用块连接器(Attacker)将各个不同的块组合成完整的句法结构树。

这样,一方面由于对不同的子问题的准确功能定位,可以独立地选用不同的语言模型和搜索策略加以分析处理;另一方面,通过在块层次上进行自底向上的块间关系分析和自顶向下的块内结构分析,可以大大提高整体分析效率,达到降低句子分析难度的目的。

语义组块的另一个用处在于浅层语法分析,即将语义角色标注的工作建立在浅层语法分析之上,不再使用句法解析树,而是利用分析出来的语法组块直接进行语义角色标注,希望利用相对更准确的组块分析结果提升语义角色标注准确率。

最后，最常用的一个领域是，语义组块可用于知识库的实体关系抽取，我们将在第 7 章详细讨论语义组块在实体关系抽取中的应用。

目前，大规模的中文熟语料库（含分词、词性标注加工的中文语料库）已经具有很大规模。这为构建更高层次的语言资源建立了坚实的基础。本文所介绍的组块规范和来源均获取自美国宾州大学的中文树库。

5.2.1 语义组块的种类

我们对组块种类的获取来自 CTB 对汉语句法短语类型的分类。CTB 将汉语句子中的短语分为如表 5.4 所示的类型。

表 5.4 CTB汉语短语类别表

标 注	英文说明	中文说明
ADJP	Adjective phrase	形容词短语
ADVP	Adverbial phrase headed by AD (adverb)	由副词开头的副词短语、状语
CLP	Classifier phrase	量词短语
CP	Clause headed by C (complementizer)	由补语引导的补语从句、关系从句
DNP	Phrase formed by “XP+DEG”	XP+DEG结构构成的短语
DP	Determiner phrase	限定词短语
DVP	Phrase formed by “XP+DEV”	XP+DEV结构构成的短语
FRAG	fragment	片段
IP	Simple clause headed by I (INFL或其曲折成分)	简单句
LCP	Phrase formed by “XP+LC”	处所词为中心语的短语
LST	List marker	用于解释说明性的列表标记短语
NP	Noun phrase	名词短语
PP	Preposition phrase	介词短语
PRN	Parenthetical	插入语
QP	Quantifier phrase	数词短语
UCP	unidentical coordination phrase	非一致性并列短语
VP	Verb phrase	动词短语

如表 5.4 所示，并非所有的短语类型都能作为语义组块。其中，IP、CP 为简单句和从句语法块，该组块本质上是一个完整的句子，这不符合 Abney 对组块的定义，因此将其删减。几乎所有的 CLP 都是 QP 的一个子集，将其与 QP 合并。FRAG 是由若干词汇

构成的一个集合，其不属于句子的范畴，不能作为组块来分析。LST 是对文本中列表说明的一种标注，不属于句内的成分，也不作为组块分析。UCP 是一种逻辑上并列的句子成分的标注，而不是具有语义组块的一种范畴，将其删除。

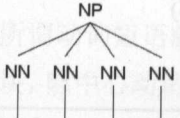
剩下的短语类型包括：ADJP、ADVP、DNP、DP、DVP、LCP、NP、PP、PRN、QP 和 VP 这几种。5.2.2 节开始对这些组块逐一详细说明。

5.2.2 细说 NP

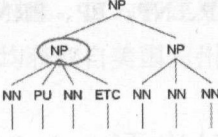
宾州树库中的名词短语是指中心词为名词所构成的短语，其语法功能相当于名词性成分，一般可以在句子中充当主语、宾语、定语等。

NP 是 NLP 组块分析中最为复杂的一种结构。从语法的角度来讲，该结构具有两种含义：一种是指按句法成分构成的短语，如组块在句子中充当主语、宾语等，这种 NP 可以增加辅助标签 NP-Sbj、NP-Obj；另一种是指知识库中的实体和属性，这种组块称为 baseNP，本节主要给出的 baseNP 的构成。

(1) 名一名复合词。连续的词性标注为 NN 词串构成的复合词被括号括起来作为 NP。对于一个 NN1 NN2...NNi 序列，尽管通常说来最后一个 NN 总是中心语，但并非前面所有的 NN 都会直接修饰最后的 NN。前面任意数量的 NN 都可能构成一个短语去修饰最后一个 NN。因为很难确定谁修饰谁，所以整个序列保留扁平化，组成最低层次的 NP。这种 NP 既可以修饰其他短语，也可以被其他短语修饰。


树状结构	例 子		
	(NP (NN 纺织) (NN 工业))	(NP (NN 工程) (NN 施工) (NN 招投标) (NN 管理) (NN 办法))	(NP (NN 下岗) (NN 分流) (NN 人员))
	(NP (NN 压锭) (NN 重组) (NN 项目))	(NP (NN 纺织品) (NN 出口) (NN 退税率))	(NP (NN 棉花) (NN 供应) (NN 办法))

(2) 词级并列结构。该名词复合结构由两部分构成，左侧为并列结构的名词短语，右侧为一个名词或名一名复合词。当它们修饰另一个名词时，就被当作短语层级的修饰语。

树状结构	例 子		
	(NP(NP (NN 改革) (CC 和) (NN 解困)) (NP (NN 方面)))	(NP (NP (NN 能源) (PU 、) (NN 通信) (ETC 等)) (NP (NN 城市) (NN 基础) (NN 设施)))	(NP (NP (NN 压锭) (NN 重组) (PU 、) (NN 兼并) (NN 破产) (PU 、) (NN 结构) (NN 调整)) (NP (NN 工作)))

(3) 由 NR (专有名词) 加上一个或多个 NN 所组成的专有名词 NR+NN 组成的专有名词。它包括如下两种情况。

① 组织或公司名称。


树状结构	例 子		
	(NP-PN (NR 深业) (NN控股))	(NP-PN (NR 一机部) (NR 上海) (NN 电器) (NN 科学) (NN 研究所)))	(NP-PN (NR 中国) (NN 机械) (NN 工业) (NN 部))

此外，如果 NR 和 NN 没有构成专有名词，其内部结构也要进行句法分析。如果组织名称包含 NR 和 NN 外的其他成分，它们的内部结构用括号进行句法分析。

(NP-PN (NP-PN (NR 深圳)) (NP (ADJP (JJ 高速)) (NP (NN 公路))) (NP (NN 股份)) (ADJP (JJ 有限)) (NP (NN 公司)))	(NP (NP-PN (NR 深) (NR 港)) (NP (NN 经济) (NN 合作) (NN 前景)))	(NP (NP-PN (NR 国民党) (NN 政府)) (NP (PU “) (NN 陪都) (PU ”)))
---	---	---

在这种情况下，NR 投射为一个 NP，该 NP 修饰由 NN 组成的 NP。

② 姓名+称谓。

树状结构	例 子		
	(NP-PN (NR 周) (NN 先生))	(NP-PN (NR 朱镕基) (NN 总理))	(NP-PN (NR 夏) (NN 教授))

注意，这与“职位+名称”不同，后者两个词是同位结构，比较如下。

树状结构	例 子
	(NP-APP (NP-PN (NR 一机部) (NR 上海) (NN 电器) (NN 科学) (NN 研究所)) (NP (NN 副所长)))

(4) 日期与地点。

① 构成日期的一连串 NT 被保留扁平化结构。

树状结构	例 子
	(NP (NT 一九九九年) (NT 四月) (NT 十九日))

② 构成地点名词的一连串 NR 也标为 NP，其内部结构也做扁平化处理。

树状结构	例 子
	(NP-PN (NR 山东省) (NR 烟台市))

5.2.3 细说 VP

所谓动词短语就是以动词为中心，与其修饰、限定或并列成分共同构成的一种语义组块，除中心动词表达的行为之外，其修饰和限定成分更明确和具体化动作的语义。在句法上，它的作用与一个单独的动词差不多，在实际分析中，要把动词短语当成一个整体。这与传统句法分析的 VP=VP+NP 的结构有显著的不同。这里给出的 VP 相当于 baseVP。

1. 复合动词短语

复合动词是指动词与动词的复合，构成一个动词的序列。两个动词之间构成动结式机构、动趋式结构、修饰语+中心语结构、并列结构等。由于在中文里缺乏区分复合词和

短语的明确标准，因此采用如下确定为复合动词的工作标准。如果一个动词序列：（1）共享相同的论元结构。（2）共享相同的体标记。（3）共享修饰语。（4）不属于明确的提升或控制结构。此处不讨论汉语中词和短语的区别问题。

如下是动词复合词的分类，并给出例子说明如何用括号方法进行句法分析。复合动词是标注员之间容易出现不一致的地方。

（1）复合动词搭配（Coordinated Verb Compounds）。两个动词之间有相同的次范畴框架，即在所出现的语境中共享论元。如果这些动词带有宾语，其标注方式可以看作如下标注方式的简略形式。

复合动词标注的结构：	例子：VP+NP
(VP (VP V1 (NP-OBJ (-NONE- *RNR*-1))) (VP V2 (NP-OBJ ...)))	(VP (VCD (VV 开发) (VV 建设) (NP-OBJ (PU “ (VE 无) (NN 人) (NN 区) (PU ”))))

更多例子：VP。

(VP (VCD (VV 回国) (VV 就职)))	(VP (VCD (VV 开发) (VV 建设)))	(VP (VCD (VV 贯彻) (VV 执行)))
(VP (VCD (VV 登记) (VV 注册)))	(VP (VCD (VV 勘探) (VV 开发)))	

（2）动结式和动趋式复合词（VRD）。通常这类复合动词由两个成分构成，其中第二个成分表示第一个成分的方向结果。举例如下。

例子：VP+NP	例子：VP	
(VP (VRD (VV 研制) (VA 成功)) (NP-OBJ (NP (NN 数字) (ADJP (JJ 程控)) (NP (NN 交换机))))	(VP (VRD (VV 下降) (VV 到)))	(VP (VRD (VV 建立) (VV 起)))
	(VP (VRD (VV 体会) (VV 到)))	(VP (VRD (VV 表达) (VV 出)))
	(VP (VRD (VV 提取) (VV 出)))	(VP (VRD (VV 武装) (VV 成为)))
	(VP (VRD (VV 联合) (VV 起来)))	(VP (VRD (VV 降) (VV 下来)))

(3) “修饰语+中心语”式复合动词 (VSB)。这类复合动词中，第一个成分应为不及物动词，而且两个成分之间没有附加语或体标记。举例如下。

例子: VP+NP	例子: VP	
(VP (VSB (VV 驱车) (VV 行程)) (QP-EXT (CD 800多) (CLP (M 公里)))	(VP (VSB (VV 拿来) (VV 支付)))	(VP (VSB (VV 仰头) (VV 望去)))
	(VP (VSB (VV 争) (VV 吃)))	

(4) “VV+VC”式复合动词 (VCP)。
例如，VP。

(VP (VCP (VV 看作)
(VC 是)))

2. 动词 (复合动词) +体标记/得

在宾州树库中，体标记 (如 “了、着、过”) 不和前面的动词括在一起，动词和体标记放在同一层级。“得”的处理方式与此类似 (这一点实在有点让人不理解)。举例如下。

例子: VP+NP	例子: VP
(VP (VV 分析) (AS 了) (NP-OBJ (CP (WHNP-1 (-NONE- *OP*)) (CP (IP (NP-SBJ (QP (CD 两) (CLP (M 次))) (NP (NN 对接))) (VP (PP-PRP (-NONE- *T*-1)) (VV 失败))) (DEC 的))) (NP (NN 原因))))	(VV 坚持) (AS 了) (VV 冒) (AS 着) (VV 意味着) (AS 着) (VV 参加) (AS 过) (VV 做) (DER 得) (VV 写) (DER 得)

如果前面是一个复合动词，该复合动词也与体标记放在同一层级。

(VCD (VV 培养) (VV 造就)) (AS 了)	
(VRD (VA 紧张) (VV 起来)) (SP 了)	
(VRD (VV 移交) (VV 给)) (AS 了)	

3. A不A, A—A, 以及变种“V不V”、“V得V”

“A不A”作为一个词层级的语类, 标为 VNV。

类型	例子: VP+NP	例子: VP
“A不A”, “A—A”	(IP (NP-PN-SBJ (NR 中国)) (VP (VNV (VV 能) (AD 不) (VV 能)) (VP (VRD (VV 研制) (VV 成功)) (NP-OBJ (NP (NN 数字) (ADJP (JJ 程控)) (NP (NN 交换机)))))))	(VNV (VV 能) (AD不) (VV 能))
“V不V”、“V得V”	(VP (VPT (VV 打) (AD 不) (VV 赢)) (NP-OBJ (DP (DT 这) (CLP (M 场))) (NP (NN 战争))))	

4. 前面三种类型相互组合, 构成更为复杂的“复合词”

如下给出了可以组合的类型。 $A(i,j)$ 表示如果前面类型为*i*, 之后可以搭配类型*j*。“?”表示结果是否可以接受有疑问。

	VCD	VRD	V-ASP	V-not-V	V-one-V
VCD	yes*	yes	yes	?	no
VRD	yes	no	yes	?	no
V-ASP	yes	no	yes**	no	no
V-not-V	yes	?	no	no	no
V-one-V	yes	no	no	no	no

*: VCD + VCD 同 VCD。

**: V-ASP + V-ASP 仅适用于 V-过-了。

举例如下。

VCD + VRD	(VRD (VCD (VV 开发) (VV 建设) (VV 出))
VCD + V-ASP	(VCD (VV 培养) (VV 起来)) (AS 了)

VRD + V-ASP	(VRD (VA 紧张) (VV 起来))
	(AS 了)
	(VRD (VV 移交) (VV 给))
	(AS 了)

5. 有连接词的并列结构

(a) 没有补足语	(b)有补足语
(VP (ADVP (AD 都)) (PP (P 与) (NP (NP-PN (NR 德国)) (NP (NN 始祖鸟)))) (VP (VA 相近) (CC 和) (VA 相似)))	(VP (ADVP (AD 不断)) (VP (VP (VV 影响) (NP-OBJ (-NONE- *RNR*-1))) (PU 、) (VP (VV 辐射) (NP-OBJ (-NONE- *RNR*-1))) (CC 和) (VP (VV 带动) (NP-OBJ-1 (DNP (NP (NP (NN 农村)) (NP (PU “) (QP (CD 两) (CLP (M 个))) (NP (NN 文明)) (PU ”)) (NP (NN 建设))) (DEG 的)) (NP (NN 发展))))))

举例如下。

(VV 继承) (CC 和) (VV 发扬)

(VV 解释) (CC 并) (VV 回答)

(VV 丰富) (CC 和) (VV 完善)

(VV 拓展) (CC 与) (VV 变革)

5.2.4 其他语义块

(1) QP: 由数量词构成的短语结构。

(QP (CD 30多) (CLP (M 名)))	(QP (CD 300多) (CLP (M 只)))	(QP (CD 5) (CLP (M 间)))
------------------------------	-------------------------------	----------------------------

(2) DP: 限定词短语, 一般用于修饰 NP 或限定 QP, 可以作为复合 NP 的子结构。

(NP (DP (DT 任何)) (NP (NN 人)))	(NP (DP (DT 全体)) (NP (NN 外交) (NN 人员)))	(NP (DP (DT 这) (QP (CD 五) (CLP (M 个)))) (NP (NN 学生)))
----------------------------------	--	--

(3) ADJP: 形容词短语, 由 JJ 投射得到, 其所修饰的名词中心语总是要先投射成一个 NP。

(NP (ADJP (JJ 若干)) (NP (NN 规定)))	(NP (ADJP (JJ 大型)) (NP (NN 会议)))	(NP (ADJP (ADVP (AD 不)) (ADJP (JJ 完全)) (NP (NN 统计)))
-------------------------------------	-------------------------------------	--

注意, 在第一个例子中, JJ+NN 组合被另一个 NP 修饰, 而在最后一个例子中, ADVP 修饰 JJ, ADJP 再修饰 NN。

(4) DNP: 由多种类型的短语加上(DEG 的)构成。它们总是出现在 NP 的上下文中。(DEG 的)除表示它前面的短语为 NP 的修饰语之外, 没有其他作用。DNP 常被看作一种复合的 NP 结构。

(NP (DNP (NP-PN (NR 张三)) (DEG 的)) (NP (NN 书)))	(NP (DNP (NP (NP-PN (NR 长江) (PU 、) (NR 嘉陵江) (NP (NN 交汇处)) (DEG 的)) (NP (NN 山丘) (NN 坡地)))	(NP (DNP (QP (CD 13) (CLP (M 年))) (DEG 的)) (NP (NN 开发) (NN 建设)))
--	--	--

(5) ADVP: 副词短语, 常用作动词的修饰语。

(IP (NP-PN-SBJ (NR 西门子)))

(VP (ADVP (AD 将)))

(ADVP (AD 努力))

(VP (VV 参与)

(NP-OBJ (DNP (NP-PN (NR 中国))

(DEG 的))

(NP (NP-PN (NR 三峡))

(NP (NN 工程)))

(NP (NN 建设))))))

(6) PP: 介词短语。

(PP (ADVP (AD 仅)) (PP (P 在) (NP (NT 一九九九年)))	(PP (P 经过) (NP (NN 努力)))	(PP (P 在) (LCP (NP (NN 山腰)) (LC 间)))
--	-----------------------------	--

(7) LCP: 处所词为中心语的短语，如其父短语是一个 PP，一般作为 PP 的子结构。独立使用时，可以看作一个不完整的 PP。

(LCP (NP (NN 传说)) (LC 中))	(LCP (NP (QP (CD 两)) (NP (NN 国))) (LC 间))
------------------------------	---

5.2.5 语义块的抽取

从本节开始，详细介绍语义组块自动识别的算法。关于语义组块识别的最常用方法是条件随机场（Conditional Random Fields，CRF）。它也是 NLP 序列标注中最有代表性的算法。在小规模的中文分词、词性标注、未登录词（命名实体）识别和语义组块，以及后面的语义角色标注中都有很广泛的应用。因为本节以算法介绍和应用为主，有关特征工程方面的问题暂不涉及，所以本节不讨论算法的性能问题。

使用 CRF 来进行语义块的识别，使用的语料一般为 Penn TreeBank 的 CTB 树库语料。前面介绍过，该语料为宾州大学开发的中文树库，标注方法为短语结构法。如果读者没有该语料，可以通过安装 NLTK 下载该语料，有关 NLTK 的安装问题，第 1 章相关内容已经详细说明了。

使用 CRF 来识别语义组块，需要通过如下三个阶段来完成。

- ❑ 将 Penn TreeBank 树库中的语料从树状结构变为序列结构。
- ❑ 使用 CRF 算法对制作好的语料进行训练，生成模型。
- ❑ 使用训练的结果，测试组块标注。

下面逐一给出每个过程所需要的执行步骤及相应的实现代码。将 Penn TreeBank 树库中的语料从树状结构变为序列结构。

(1) 宾州树库的源文件格式样例与转换。

((IP-HLN (NP-SBJ (NP-PN (NR 上海)

(NR 浦东))

(NP (NN 开发)

(CC 与)

(NN 法制)

(NN 建设)))

(VP (VV 同步)))

如果不对组块的定义有特殊要求，则推荐使用国外专家编写的基于 Perl 的小程序：ChunkLinkCTB。用户可以从 <https://github.com/rainarch/ChunkLinkCTB>（或 <http://www.threedweb.cn/thread-1588-1-1.html>）下载。程序需要运行在 Perl 环境下，如果是 Windows 用户则需要安装 perl 程序包（下载地址：<http://strawberryperl.com/>）。

在命令行中输入执行指令，内容如下。

```
perl chunklinkctb.pl -fhHct 源文件.mrg > 目标文件.chunk
```

截图如下。

```
D:\workSpace\NLPBook\chapter5\perl>perl chunklinkctb.pl -fhHct chtb
Use of uninitialized value within @args in string eq at D:/Strawber
1
405 words processed
```

(2) 转换后的序列标注文件样例。

序列标注转换样例表如表 5.5 所示。

表 5.5 序列标注转换样例表

文件id (file_id)	句子id (sent_id)	词汇id (word_id)	序列标签 (iob_inner)	词性标签 (pos)	词汇 (word)
1	1	0	B-NP	NR	上海
1	1	1	I-NP	NR	浦东
1	1	2	B-NP	NN	开发
1	1	3	I-NP	CC	与
1	1	4	I-NP	NN	法制
1	1	5	I-NP	NN	建设
1	1	6	B-VP	VV	同步

表 5.5 中的前三列主要是为了索引原始树库的索引记录，在序列标注阶段都没用。后三列需要根据 CRF 算法训练文件的要求改变一下顺序，改变后的结果如图 5.2 所示(因为实现程序比较简单，略)。

上海	NR	B-NP
浦东	NR	I-NP
开发	NN	B-NP
与	CC	I-NP
法制	NN	I-NP
建设	NN	I-NP
同步	VV	B-VP
新华社	NN	O
上海	NR	O
二月	NT	O
十日	NT	O
电	NN	O

图 5.2 CRF 用序列标注训练语料样例表

注意，图 5.2 最终生成的训练样本为一个纯文本文件，字符集编码应为 UTF-8 格式。两列之间用空格 (Space) 键或 Tab 键 (\t) 隔开，行尾换行符为 “\n”，句子与句子之间用空行相隔。在训练文件的末尾，最好不要留有多余的空行。

(3) 序列标签说明。

ChunkLinkCTB 得到的序列标签共分为两部分，用 “-” 分割，前面部分称为 IOB 表示法 (也称作 BIO 表示法)，后面部分标签对应所有的语义组块名称。

IOB 表示法常用于序列标注的各种算法中。“B”代表当前词是一个组块的开始，“I”代表当前词在一个组块中，“O”代表当前词不在任意一个组块中。ChunkLinkCTB 默认使用此方法进行标注。如果读者希望达到更高的精度，我们还推荐使用 IOB 的一个变种——start/end 表示法，在中文分词中最常使用的就是这种方法。Start/end 的优势是可对序列表达得更为细致，它共有 5 种符号：B、I、E、O 和 S。这 5 种符号表示的意义如下。

- ☐ B: 当前词是一个组块的开始。
- ☐ I: 当前词在一个组块内部。
- ☐ E: 当前词是一个组块的终结。
- ☐ O: 当前词不在任意一个组块中。
- ☐ S: 当前词是一个组块，该组块只有一个词。

上例如转换为 start/end 表示法则为表 5.6 的形式，两种标注的转换并不复杂，有兴趣的读者可以尝试自己实现一下。

表 5.6 IOB和start/end表示法对照样例表

词汇 (Word)	词性标签 (Pos)	序列标签 (IOB)	序列标签 (start/end)
上海	NR	B-NP	B-NP
浦东	NR	I-NP	E-NP
开发	NN	B-NP	B-NP
与	CC	I-NP	I-NP
法制	NN	I-NP	I-NP
建设	NN	I-NP	E-NP
同步	VV	B-VP	S-VP

5.2.6 CRF 的使用

下面使用 CRF 算法对制作好的语料进行训练，生成模型。

1. CRF 的工具包介绍

CRF 算法比较复杂，对训练精度和速度要求都很高，实验级别的学习和应用不建议使用基于脚本语言的工具包。目前，C/C++的实现框架很多。其中，最著名的有如下两个包。

- ❑ CRF++开源工具包，项目地址：<https://taku910.github.io/crffpp/>。该工具包用 C++ 语言编写，算法实现了 LBFSG 参数估计办法，以及多线程的训练过程。它不仅训练速度快、精度高，而且代码简洁易懂，便于学习。它是开发最早、使用最广泛的软件包之一。本书使用的就是该软件包。
- ❑ CRFsuite，项目地址：<http://www.chokkan.org/software/crfsuite/>。它也实现了 LBFSG 参数估计办法。该工具包的优势是代码用 C 语言写成，源代码跨平台，兼容 Linux 和 Windows，并与外部脚本程序结合得很好。例如，CRFsuite 提供了 Python 的扩展，可以通过 Python 直接调用算法，非常适合那些不需要了解细节的外部应用。

2. CRF++的项目安装和介绍

打开 CRF++项目的主页（<https://taku910.github.io/crffpp/#download>），CRF++下载界面如图 5.3 所示。

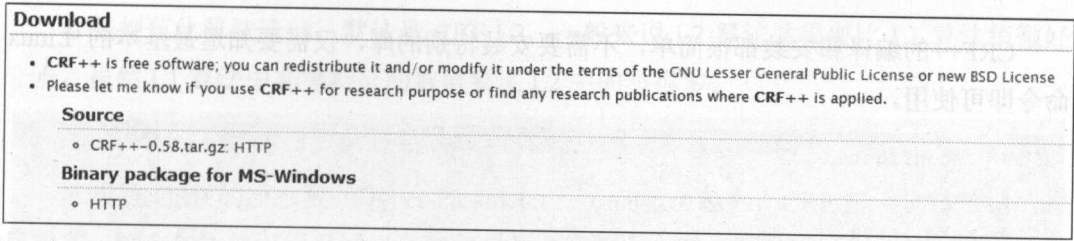


图 5.3 CRF++下载界面

CRF++源码下载界面如图 5.4 所示，目前最新版为 0.58 版。

CRF++-0.56.tar.gz	2015/3/14
CRF++-0.56.zip	2015/3/14
CRF++-0.57.tar.gz	2015/3/14
CRF++-0.57.zip	2015/3/14
CRF++-0.58.tar.gz	2015/3/14
CRF++-0.58.zip	2015/3/14

图 5.4 CRF++源码下载界面

Windows 用户可以下载编译后的动态链接库版本。因为涉及源码的调试，我们使用 Linux 环境下的 CRF++源码，如图 5.5 所示。

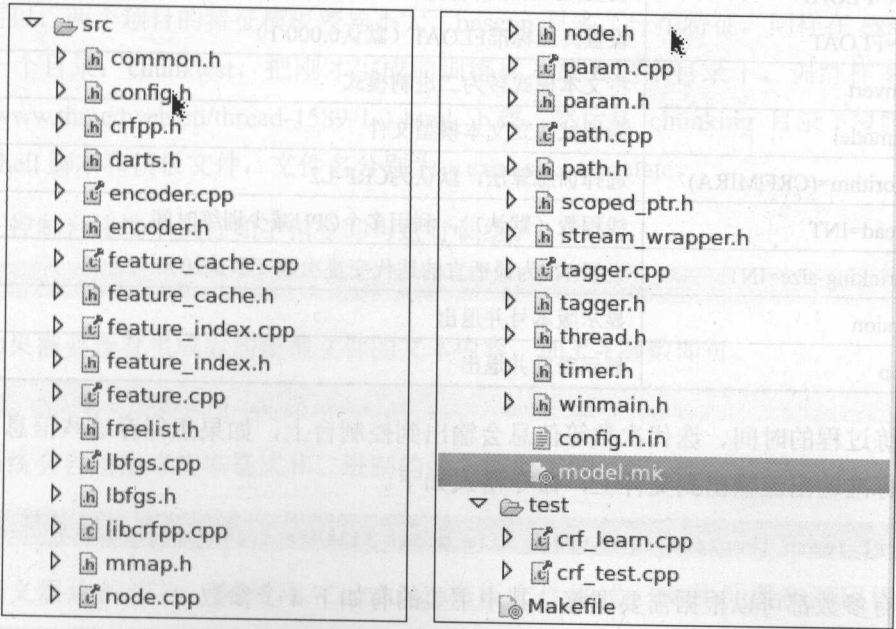


图 5.5 CRF++源码界面

CRF++的编译和安装都很简单，不需要安装特别的库，仅需要知道最基本的 Linux 命令即可使用。

```
# ./configure
# make
# make install
# ldconfig
```

3. 使用 CRF 算法对制作好的语料进行训练，生成模型

1) 训练指令与可选参数

训练指令示例如下。

```
../crf_learn -[可选参数] template train.data model
```

可选参数表如表 5.7 所示。

表 5.7 可选参数表

参 数	说 明
-f, --freq=INT	使用属性的出现次数不少于INT（默认为1）
-m, --maxiter=INT	设置INT为LBFGS的最大迭代次数（默认10k）
-c, --cost=FLOAT	设置FLOAT为代价参数，过大会过度拟合（默认1.0）
-e, --eta=FLOAT	设置终止标准FLOAT（默认0.000 1）
-C, --convert	将文本模式转为二进制模式
-t, --textmodel	为调试建立文本模型文件
-a, --algorithm=(CRF MIRA)	选择训练算法，默认为CRF-L2
-p, --thread=INT	线程数（默认1），利用多个CPU减少训练时间
-H, --shrinking-size=INT	设置INT为最适宜的迭代变量次数（默认20）
-v, --version	显示版本号并退出
-h, --help	显示帮助并退出

训练过程的时间、迭代次数等信息会输出到控制台上，如果想保存这些信息，可以将这些标准输出流输出到文件上，命令格式如下。

```
% crf_learn template_file train_file model_file >> train_info_file
```

所有参数都可以根据需要调整，其中主要的有如下 4 个参数。

```
-a CRF-L2 or CRF-L1
```


-a 是规范化算法选择。默认是 CRF-L2。一般来说 L2 算法效果要比 L1 算法稍微好一点，虽然 L1 算法中非零特征的数值要比 L2 中的小很多。

```
-c float
```

-c 设置 CRF 的超参数 (Hyper-Parameter)。c 的数值越大，CRF 拟合训练数据的程度越高。这个参数可以调整过拟合和欠拟合之间的平衡度，可以通过交叉验证等方法寻找较优值。

```
-f NUM
```

-f 设置特征的 cut-off threshold。CRF++ 使用训练数据中至少 NUM 次出现的特征。默认值为 1。当使用 CRF++ 到大规模数据时，只出现一次的特征可能会有几百万个，这个选项就会在这样的情况下起到作用。

```
-p NUM
```

-p NUM 是线程数量，如果电脑有多核 CPU，那么可以通过多线程提升训练速度。

2) 示例和运行脚本

在 CRF++ 的 example 目录中 (路径: CRF++-0.58/example/) 有几个示例项目，分别为: basenp、chunking、japaneseNE、seg。如果希望使用执行中文分词，可以参照 seg 下的示例。我们的任务是组块标注，参照 basenp 和 chunking 两个案例的特征文件和执行脚本都可以。两个项目的特征模板差异不大，basenp 只多了一个特征。同样在 example 下新建一个目录: chunktest，把刚才生成的训练样本放到这个目录下，训练样本可以从 <http://www.threedweb.cn/thread-1589-1-1.html> 下载。然后从 chunking 目录下复制得到执行的 shell 脚本和模板文件，文件名分别为: exec.sh、template。

在控制台终端中执行如下指令即可进行训练。

```
% ../../crf_learn template chtb_utf8.chunk chunkmodel
```

如果需要查看生成后的模型文件的文本内容，加上 -t 参数即可。

```
% ../../crf_learn -t template chtb_utf8.chunk chunkmodel
```

系统会自动生成文本格式和二进制的两个模型文件。

3) 特征模板及其说明

语义组块的识别一般使用 Chunking 模板文件格式，文件位置在源码根目录的 example/chunking/ template。

```

# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]

U10:%x[-2,1]
U11:%x[-1,1]
U12:%x[0,1]
U13:%x[1,1]
U14:%x[2,1]
U15:%x[-2,1]/%x[-1,1]
U16:%x[-1,1]/%x[0,1]
U17:%x[0,1]/%x[1,1]
U18:%x[1,1]/%x[2,1]

U20:%x[-2,1]/%x[-1,1]/%x[0,1]
U21:%x[-1,1]/%x[0,1]/%x[1,1]
U22:%x[0,1]/%x[1,1]/%x[2,1]

# Bigram
B

```

由上述可知, CRF++的模板类型有如下两种: Unigram Template, 也称为一元特征模板; Bigram Template, 也称为二元特征模板。下面给出两种模板的说明。

第一种是一元特征模板 (Unigram Template): 模板的第一个字符是 U, 这是用于描述一元特征模板。当给出一个模板 "U00:%x[-2,0]" 时, CRF 会自动生成一个特征函数集合 (func1 ... funcN), 内容如下。

```

func1 = if (output = B and feature="U00:Tulsa") return 1 else return 0
func2 = if (output = I and feature="U00:Tulsa") return 1 else return 0
func3 = if (output = O and feature="U00:Tulsa") return 1 else return 0
....
funcX1 = if (output = B and feature="U01:'s") return 1 else return 0
funcX2 = if (output = I and feature="U01:'s") return 1 else return 0
funcX3 = if (output = O and feature="U01:'s") return 1 else return 0
...

```

一元模型生成的特征函数的个数总数为 $L \times N$ 。其中， L 是类别标签数， N 是根据给定的模板扩展出的一元特征数。它构成了一元特征函数矩阵，矩阵的行是模板扩展出的一元特征函数，列为类别标签。

已知如表 5.8 所示的输入数据。

表 5.8 训练语料样本

上海	NR	B-NP	<< 当前Token位置
浦东	NR	I-NP	
开发	NN	B-NP	
与	CC	I-NP	
法制	NN	I-NP	
建设	NN	I-NP	
同步	VV	B-VP	

按照特征模板，组块的训练样本一共有三列，前两列为样本取值，最后一列为分类标签，每一行称为一个 Token。模板参数也需要两列，其坐标[0,0]代表两个训练样本的两列，第一个表示词汇列，第二个表示词性列。数值 0 是当前位置，-1 是当前词的前一个位置，1 是当前词的后一个位置。模板及其对应特征说明如表 5.9 所示。

表 5.9 模板及其对应特征说明

模 板	对应特征
%x[0,0]	与
%x[0,1]	CC
%x[-1,0]	开发
%x[-2,1]	NR
%x[0,0]/%x[0,1]	与/ CC
ABC%x[0,1]123	ABCCC123

第二种是二元特征模板（Bigram Template）：模板的第一个字符是 B。使用这个模板，系统将自动产生当前输出标签与前一个输出标签的二元组合（Bigram）。产生的可区分的特征的总数是 $L_p \times L_c \times N$ 。其中， L_p 是前一个输出类别标签数； L_c 是当前输出类别标签数（ $L_p = L_c$ ）； N 是这个模板产生的一元特征数。它构成了二元特征函数矩阵，构成矩阵的行和列都是类别标签，但行表示前一个类别标签（ L_p ），列表示当前的类别标签（ L_c ）。

当类别数很大时，这种类型会产生许多互不相同的特征，特征数的激增会导致训

练和测试的效率都很低下。该类模板因为训练效率较低，一般很少使用。在特征模板中只使用“B”，表示二元特征矩阵是仅由当前的前一个输出 L_p 标签和当前输出标签 L_c 构成。

4) 训练输出日志

```
CRF++: Yet Another CRF Tool Kit
Copyright (C) 2005-2013 Taku Kudo, All rights reserved.

reading training data:
Done!0.01 s

Number of sentences: 11
Number of features: 52624
Number of thread(s): 2
Freq:          1
eta:           0.00010
C:             10.00000
shrinking size: 20
iter=0 terr=0.99506 serr=1.00000 act=52624 obj=1122.89843 diff=1.00000
iter=1 terr=0.32593 serr=1.00000 act=52624 obj=881.06755 diff=0.21536
iter=2 terr=0.22716 serr=1.00000 act=52624 obj=247.73335 diff=0.71883
...
iter=23 terr=0.00000 serr=0.00000 act=52624 obj=18.64905 diff=0.00017
iter=24 terr=0.00000 serr=0.00000 act=52624 obj=18.64876 diff=0.00002
iter=25 terr=0.00000 serr=0.00000 act=52624 obj=18.64738 diff=0.00007
iter=26 terr=0.00000 serr=0.00000 act=52624 obj=18.64715 diff=0.00001

Done!0.70 s
```

说明如下。

- ☐ iter: 迭代次数。
- ☐ terr: 标签 (tag) 错误率。
- ☐ serr: 句子 (sentence) 错误率。
- ☐ obj: 当前对象值，该值收敛于一个固定的值则停止迭代。
- ☐ diff: 与上次对象值的相对差异。

5) 文本模型文件说明

最后，给出训练完成后，生成文本形式的模型文件说明，内容如下。

□ 文件头。

version: 100	左侧为样本输出的文件头。
cost-factor: 1	version: 模型的版本。
maxid: 52224	cost-factor通过-c参数指定，这里是默认值。
xsize: 2	maxid: CRF矩阵的元素总数量。
	xsize是特征维数，也就是训练语料列数-1。

□ 标签和模板。

接下来的两块为训练样本中使用的标签和模板，模板前文已经讲过，这里就不重复了。仅给出统计出的全部标签：B-ADJP、B-ADVP、B-DNP、B-DP、B-DVP、B-LCP、B-NP、B-PP、B-QP、B-VP、I-ADVP、I-DP、I-NP、I-QP、I-VP 和 O。从训练集中得到的标签数一共为 16 个。

□ 特征函数和权值。

截取的部分特征函数列表，内容如下。

304 U00: _B-1
0 U00: _B-2
17664 U00: "
20032 U00: "
4400 U00: 、
11424 U00: —
30656 U00: 一亿多
8480 U00: 七十一
592 U00: 上海

- “_B-1”表示句子第一个单词前面的一个单词，_B+1 表示末尾后面的一个单词，依次类推。
- “U00:上海”：冒号前为一元特征编码，冒号后为一元特征对应的词汇。通过模板和训练语料学习出的全部一元特征函数一共有 3 249 个，即 3 249 行。
- 一元特征前面的序号，如“U00:上海”前的序号为 592。因为标签总数为 16 个，592 除以 16 得到结果为 37，表示“U00:上海”位于一元特征函数矩阵的第 37 行。
- 一元特征函数矩阵的总元素数为：3 249×16=51 984

截取的部分矩阵权值列表，内容如下。

```
-0.0100499553958394
0.3220849114124381
-0.0094501499990385
-0.0079356846560915
-0.0493756198456124
-0.0264509527150632
-0.0243607765220631
-0.0260220669646972
-0.0320371687631909
```

- 计算得到的矩阵中每个元素对应权值。矩阵包括一元特征函数矩阵和二元特征函数矩阵。
- 由于对矩阵中的每个元素做了顺序存储，所以两个矩阵都表示为列的形式。
- 它包括一元特征函数矩阵和二元特征函数矩阵的全部元素值。

二元特征矩阵的总元素数为：16×15=240。

全部元素共有 52 224 个，也就是 maxid 的值。

4. 测试训练结果

执行命令行如下。

```
% crf_test -m chunkmodel test
```

测试后的训练样本如图 5.6 所示。

```
[root@localhost chunktest]# ../../crf_test -m chunkmodel test
上海      NR      B-NP
浦东      NR      I-NP
开发      NN      B-NP
与         CC      I-NP
法制      NN      I-NP
建设      NN      I-NP
同步      VV      B-VP
```

图 5.6 测试后的训练样本

读者可以根据训练文件，自己编写一个测试文件来验证执行结果。这里不再演示。

5.3 命名实体识别

第 3 章已简单探讨过命名实体的实现，当时是作为中文分词的一部分来介绍的。本节将全面、详细地介绍命名实体的概念、模块架构及对应的算法策略。

5.3.1 命名实体

命名实体识别 (Named Entity Recognition, NER), 又称为“专名识别”, 它的主要任务是对于一篇待处理文本, 识别出其中出现的人名 (Person)、地名 (Location)、组织机构名 (Organization)、日期 (Data)、时间 (Time)、百分数 (Percentage)、货币 (Monetary Value) 这 7 类命名实体。其中, 人名、地名、组织机构名的识别是最难、也是最重要的三类。本节将主要讲述这三类命名实体的识别。从语言分析的全过程来看, 命名实体识别属于中文分词中未登录词识别的范畴。在涉及未登录词识别的所有问题中, 命名实体是未登录词中数量最多、识别难度最大、对分词效果影响最大的问题。命名实体识别技术是信息抽取、信息检索、机器翻译、问答系统等多种自然语言处理技术必不可少的组成部分。

1. 中文人名的特征

中文人名一般由姓氏和名字两部分构成, 其构成方式包括: 单字姓+单字或双字名、复姓 (如上官、欧阳等)+单字或双字名、双姓 (父母姓氏)+单字或双字名、夫姓+父姓+单字或双字名 (如陈方安生、范徐丽泰等)、单姓+三字名, 等等。

虽然《中华姓氏大辞典》显示, 中国古今各民族用汉字记录的单姓 6 931 个、双字姓 4 329 个、三字姓 1 615 个、四字姓 569 个、五字姓 96 个、六字姓 22 个、七字姓 7 个、八字姓 3 个、九字姓 7 个 (如龔邯汕寺武穆云籍鞫)、十字姓 1 个 (伙尔川扎木苏他尔只多), 但姓名汉字的个数以 2~4 字占绝大多数, 而当今仍然使用、活跃的中文姓氏大概只有 1 000 个。据统计, 常用的姓氏分布也很不均匀且相对集中。“王、李、刘、张、陈”这 5 大姓就占了姓名样本数的 29.1%, 前 18 个姓占 50.3%, 前 181 个姓占 90.3%, 前 586 个姓占 98.6%, 其余姓氏仅占不到 1.5%。常用姓氏文件可从 <http://www.threedweb.cn/thread-1585-1-1.html> 下载。

名字用字分布较姓氏用字要平缓、分散。共得到 3 679 个名字用字, 频率最高的前 17 个字的覆盖率为 10.5%, 前 80 个字为 30.3%, 前 207 个字为 50.3%, 前 1 122 个字为 90.4%。名字用字涉及范围很广。从所属的词类看, 不仅有实词, 还有各类虚词。如副词“常、太、必、非、更、也、级、又、皆”等, 介词“以、向、从、于、把”, 连词“而、虽、且、与”等。从感情色彩看, 多使用褒义字和中性字, 但也出现了一些贬义字或不太文雅的字, 如“狼、恶、悲、暴、虫”等。

值得注意的是, 某些汉字既可用作姓氏, 又可用作名字用字。如“林、方、金、江、万、颜、童、柳”等。

2. 中文地名的特征

较之人名相比,地名更像一个闭集。绝大部分地名都可以通过相关的资料覆盖,如《中国地名录》、《地理词典》、《地物名称》词典集(可从 <http://www.threedweb.cn/forum-73-1.html> 下载)。而且,地名结尾经常有地名特征词出现,如省、市、县、乡、村、山等。多个地名常通过一些连接词或者连接符号一起出现,如“/吉林省/四平市/梨树县/梨树镇/霍家店村”。这样的地名大多表示行政地区的地名,这些对地名识别来讲都是有利的信息。上述都是地名识别比较容易的地方。

同时,地名识别也有其困难之处。

第一,据统计,中文地名用词一方面比较自由、分散,地名录中共享汉字 3 685 个;另一方面,中文地名用词又有相对集中的覆盖能力。但有时候,地名特征词出现的情况比较复杂:既可以作为普通用词出现,而不是真正的、具体的地名,如小山村、大都市,又可以出现在地名其他位置或作为地名的前部词,如三门峡市、张家界市等。

第二,不像中文姓名那样,地名长度没有严格限制。在真实文本中,经常会有地名简称出现。如“峨嵋山”常写作“峨嵋”,因为在武侠作品中它还代表一个武功门派,所以识别中常产生歧义。地名还可能和专名发生冲突。例如,作为其他命名实体的一部分,如“大连市机械厂”;作为组织机构名的一部分,如“海宁市长安邮电局”,切分成“海宁市/长安/邮电局”。这里长安容易被误切分为一个地名的古称。

第三,地名用词的情况非常复杂。地名常常与介词、动词、方位词之类的指示词连用,这些指示词对地名识别能起到标志作用,但有些指示词也可以作为地名组成部分,如“上甘岭”、“来复乡”等。还有一些地名,每一个单字均为高频单字词,如“西/直/门”、“白/家/塔”。这些字如果连用则作为地名用词的可能性非常大,但如果单个出现或部分出现,则作为非地名成分出现的次数也很多。

第四,多字词可以在地名不同的位置出现,可以在地名首部出现,也可以在地名中部出现。同时,相对于单字词来讲,地名词典中的多字词统计的信息不充分,对多字词的判断也是地名识别中的一个难点。

上述情况都增加了地名识别难度。综上所述,地名识别同样需要解决交叉歧义、边界模糊等问题。

3. 组织结构特征

较之前两种命名实体的识别,组织结构的识别更加困难。上述各种难点不同程度地出现在组织机构名识别中。此外,组织机构名的识别还有自身的问题。首先,中文组织

机构名词常常表示为两种结构：全称和简称。“联想”、“方正”都是机构的简称。有时，同一机构的全称有不同的简称。例如，“上海交通大学”可以简称为“上海交大”或者“交大”。再如，“人民代表大会”的简称为“人大”或“人代会”，而“人民大学”也简称为“人大”。大量的机构简称的出现，使得机构名的识别变得更加困难。

其次，中文机构名的长度更加不稳定。少的仅仅有两个字，如“宝钢”，长的几乎比一个句子还要长，例如“中国人民政治协商会议第八届全国委员会常务委员会”。这给中文的命名实体边界的确定造成了很大的困难。

最后，组织机构名含有复杂的内部结构，通常是其他的命名实体。在这些命名实体中，地名占很大的比例，人名等也占相当一部分的比例。这些成分都制约了组织机构名的识别。

5.3.2 分词架构与专名词典

命名实体识别中遇到各种各样难以克服的困难，人们为此也想出了很多解决的办法。人们提出了各种不同的分词架构，期望通过分词架构的设计或者引入外部资源（专名词典）来提高命名实体识别精度或者减小命名实体识别对整体分词结果的干扰。

一个大型分词系统的简单架构如图 5.7 所示。

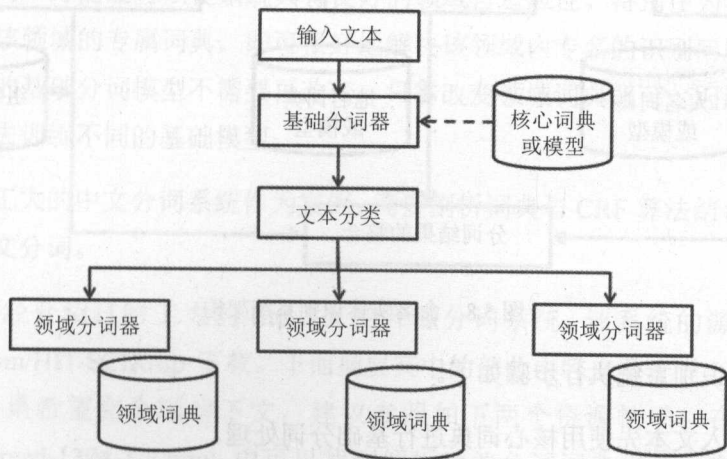


图 5.7 一个大型分词系统的简单架构

领域分词系统的执行步骤如下。

- (1) 将输入文本先使用核心词典进行基础分词处理。
- (2) 对分词结果的文本再按所属领域计算文本所属的分类。

(3) 根据文本所属的领域不同, 执行使用领域专属的分词器, 再对领域内的专名进行分词。

上述架构的特点, 是将分词器针对不同领域做了优化, 不同领域的文本可以使用本领域内专门的外部词典进行识别。专业化的细分势必带来更高的识别精度, 这种策略针对领域专属的专名往往能够达到更好的分词效果。

其中, 一种设计思路是将命名实体任务从中文分词的基础模块中分离出来, 命名实体作为一个独立的模块位于基础分词步骤之后。在 HITLp 3.3 (哈工大开源语言包 <https://github.com/HIT-SCIR>) 和 Stanford NLP 中都把命名实体识别作为一个单独的模块来处理, 独立于分词流程之外。这样做的好处是命名实体模块可以具有更丰富的结构。比如, 人名识别、地名识别和组织机构名识别都作为单独的子模块来处理。命名实体识别系统架构如图 5.8 所示。

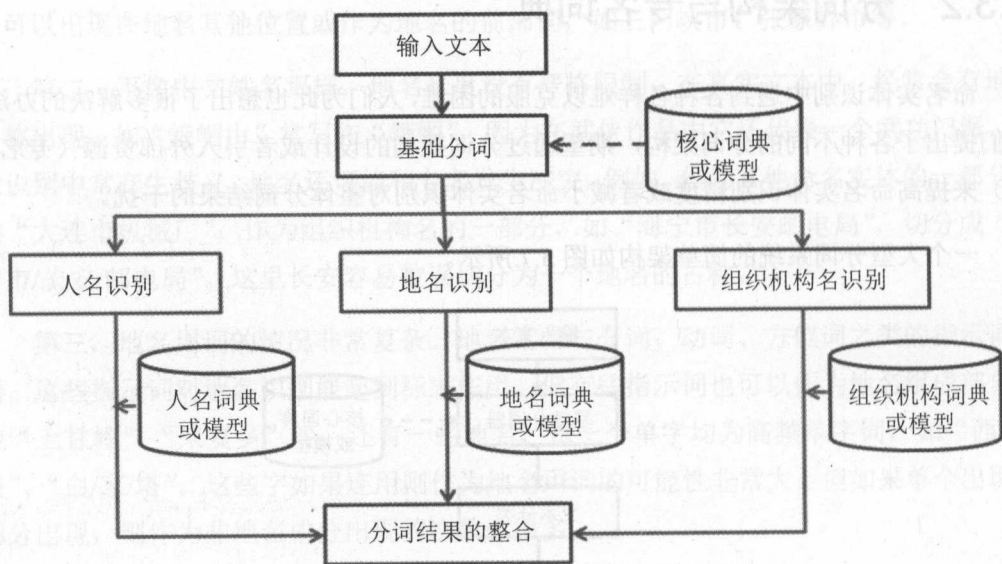


图 5.8 命名实体识别系统架构

命名实体识别系统执行步骤如下。

(1) 将输入文本先使用核心词典进行基础分词处理。

(2) 再对分词结果进行命名实体识别。命名实体识别分为如下三个步骤。

① 使用专门的模型对分词结果进行人名识别。

② 使用专门的模型或词典对分词结果进行地名识别。

③ 使用专门的模型或词典对分词结果进行组织机构名识别。

(3) 结果整合。将三种结果合并到基础分词的结果中作为最终的结果输出。常用的合并方式可以直接覆盖，好一些的使用最短路径算法。

上述架构的好处在于，对于不同类型的命名实体，可以使用专门的算法和专门的外部词典进行识别，专业化的算法和语料资源对专门的问题势必达到更高的精度。系统再将不同类型的识别结果通过一定的算法整合成最终的输出结果。这样所达到的分词结果往往更好。

实践证明，标准的 CRF 算法即可解决大多数人名识别的问题；而地名的词汇更像一个闭集，丰富的外部词典对地名的识别具有很重要的作用。结合词典的 CRF 算法是 CRF 的一个变种，通常在地名识别上会达到更好的效果。对于组织机构识别国际上最通用的是 semi-CRF 算法，国内的层叠 HMM 算法也比较成功，对于专业化比较强的命名实体识别可以达到比较好的效果。

5.3.3 算法的策略——词典与统计相结合

前文中不同程度地提到了外部词典的引入。引入外部词典对命名实体识别有着很大的用处，而且外部词典的获得所需要的代价远远小于为相关领域标注分词语料所需要的代价。如果现有的分词方法能充分合理地利用外部词典，一方面可以提高句子中命名实体的准确率，另一方面还可以使系统具有良好的领域自适应性。特别在为特定领域分词时，只需加载该领域的专属词典，即可很好地解决该领域内专名的识别问题。当领域改变之后，原有的基础分词模型不需要再改变，只需改变领域词典即可，因此不需要针对不同领域重新去训练不同的基础模型。

本节以哈工大的中文分词系统作为案例，简要剖析词典与 CRF 算法结合如何实现领域自适应的中文分词。

第 1 章已经介绍过哈工大的 Ltp 3.X 的开源分词系统。该系统的源代码可以从 <https://github.com/HIT-SCIR/ltp> 下载。下面摘要其中的部分内容来看一下 Ltp 3.2 系统的相关实现。如果希望完全理解下文，建议参照如下两个资源帖子：在 <http://www.threedweb.cn/thread-1394-1-2.html> 中可以找到解析后的分词词典文件；在 <http://www.threedweb.cn/thread-1396-1-1.html> 中可以找到有关中文分词部分的源码解析文章。

下载和解压解析后的分词词典文件之后，可以看到词典文件一共分为如下 4 类文件。

- ☐ internal_lexicon.dat: 内部词典文件。
- ☐ labels.dat: 标签表。

❑ space0~14: 特征函数表, 一共 15 个文件。

❑ param: 特征函数的权值表。

对上述解析出的数据表分别介绍如下。

1. 标注、特征模板、语言模型

Ltp 3.2 中文分词系统是基于 CRF 算法的中文分词系统, labels.dat 指明其标注方法使用的是 start/end 表示法。

❑ B: 当前字是一个词的开始。

❑ I: 当前字在一个词内部。

❑ E: 当前字是一个词的终结。

❑ S: 当前字就是一个词, 该词是独字词。

模板训练出的 CRF 语言模型保存在 15 个词典文件中, 所取的文件名分别为: space0~space14。从 space0 到 space14 特征词典的词条序列号呈跨文件的递增形式, 15 个文件共计收录了 1 618 355 个语言模型特征。词典文件词条中还有几个断句标志符, 它们分别为 BOS = "_bos_"; EOS = "_eos_"; BOT = "_bot_"; EOT = "_eot_". 这 4 个标志符在词典装入模板时用于区分句首与句尾的边界。

通过源代码的分析, 可以得到 HIT-Ltp 3.2 中文分词的特征模板 (见表 5.10)。

表 5.10 HIT-Ltp 3.2 中文分词的特征模板

词典文件名	词条标识符	含 义
CRF算法中使用的特征函数模板		
space0	1={c-2}	当前字与向前两个字
space1	2={c-1}	当前字与前一个字
space2	3={c-0}	当前字本身在语料库中
space3	4={c+1}	当前字向后一个字
space4	5={c+2}	当前字向后两个字
space5	6={c-2}-{c-1}	当前字的前两个字
space6	7={c-1}-{c-0}	当前字与前一个字
space7	8={c-0}-{c+1}	当前字与后一个字
space8	9={c+1}-{c+2}	当前字的后两个字
space9	14={ct-1}	判断当前字符的前一个字符的Chartype类型
space10	15={ct-0}	判断当前字符的Chartype类型
space11	16={ct+1}	判断当前字符的后一个字符的Chartype类型

续表

词典文件名	词条标识符	含 义
内部词典匹配结果的特征模板		
space12	17={lex1}	表示当前字与其后的 n 个字在内部词典中是否成词，0表示未成词，如果是2，则是二字词，该字是二字词的词首。其他取值依次类推
space13	18={lex2}	表示当前字与其前的 n 个字在内部词典中是否成词，0表示未成词，如果是2，则是二字词，该字是二字词的词尾。其他取值依次类推
space14	19={lex3}	表示当前字与其前、后的 n 个字在内部词典中是否成词，0表示未成词，如果是4，则是四字词，该字是四字词的词中。其他取值依次类推

该算法使用的特征模板与 CRF++大同小异，其中，c 表示原子字符；ct 表示原子字符类型（Chartype）。但也有特殊之处，在内部词典匹配结果的特征模板中，lex1~lex3 表示在内部词典查询到词的位置标识。下面给出一个例子。

如果测试样本中包含“首都”这个词，而“首都”一词可以从内部词典中查到。在用 CRF 识别时，“首”字的内部词典特征为，{lex1}=2，{lex2}=0，{lex3}=0；“都”字的内部词典特征为，{lex1}=0，{lex2}=2，{lex3}=0。通过内部词典特征模板的选项决定了从词典查询到的词汇在分词结果中的优先地位。

最后，所有的矩阵概率值都被放到了 Param 表中。其中，_W[i]是概率值；_W_sum[i]是频率值。Param 表的长度由如下两部分构成。

- (1) 1 618 351（词典总词条）×4（labels 数）=6 473 404。表示 space 中的词属于每个 labels 标签的概率（频率）。也是 CRF 中一元特征函数矩阵的总元素数。
- (2) 4×4=16 表示语料库中两两标签搭配的概率。也是 CRF 中二元特征函数矩阵的总元素数。

这两项相加后值为 6 473 404+16=6 473 420，即 Parameter 维度的总行数。

2. 内部词典

需要重点讲解的部分是，该算法提供了一本简易的内部词典，就是前面名为 internal_lexicon.dat 的文件，部分输出结果如图 5.9 所示。

24713	0x7f4d934eb8bc	("自己", 1365363042, 1, 1, -1)
24714	0x7f4d934eb8d0	("意志", 1479999972, 1, 1, -1)
24715	0x7f4d934eb8e4	("信念", 3797777261, 1, 1, -1)
24716	0x7f4d934eb8f8	("当前", 1200425364, 1, 1, 4747)
24717	0x7f4d934eb90c	("我们", 1898291594, 1, 1, -1)
24718	0x7f4d934eb920	("处在", 2878430066, 1, 1, 136)
24719	0x7f4d934eb934	("转型", 2691715439, 1, 1, -1)
24720	0x7f4d934eb948	("时期", 3497336004, 1, 1, 3544)
24721	0x7f4d934eb95c	("矛盾", 1927209829, 1, 1, -1)
▶ 24722	0x7f4d934eb970	("比较", 1664811906, 1, 1, 3937)
24723	0x7f4d934eb984	("党委", 3970192644, 1, 1, -1)
24724	0x7f4d934eb998	("官僚主义", 1194309506, 1, 1, -1)
24725	0x7f4d934eb9ac	("腐败", 1130484702, 1, 1, -1)
24726	0x7f4d934eb9c0	("其他", 3999032893, 1, 1, -1)
24727	0x7f4d934eb9d4	("现象", 179831727, 1, 1, -1)

图 5.9 部分输出结果

图 5.9 中，该词典一共收录了 8 625 个词，是一本简易的内部词典。从该词典中查到的词汇在分词阶段具有更高的优先级。

下面看一下词典的具体内容，以词条“比较”为例，输出的字符串为：0x7f4d934eb970 ("比较", 1664811906, 1, 1, 3937)。其节点结构如表 5.11 所示。

表 5.11 节点结构

0x7f4d934eb970	Hash节点的地址
比较	词条字符串
1664811906	词条字符串的hash值
1	占位符——原为词条序号
1	词频
3937	HashMap桶内下一词的词条序号，-1为无节点

在分词阶段优先选取内部词典中的词汇，再对剩下的字串进行分词。内部词典定义词频默认全部为 1。系统还提供外部的扩展词典文件接口：External_lexicon，在实际应用中，可以根据需要导入专名词典文件。

3. 分词流程

流程说明如下。

(1) 首先通过原子切分，将字符串转换为字符列表，这个阶段与 ICTCLAS 的分词方法相似。

(2) 与内部词典（或用户导入的外部词典）进行最大匹配，根据匹配结果给出成词的特征函数。

- (3) 将字符串与匹配到的词典词汇共同匹配特征模板，包括 CRF 的标准模板和内部词典匹配结果特征模板。
- (4) 根据匹配的模板，查询和计算模板的概率。
- (5) 建立词网，使用 Viterbi 算法解码。
- (6) 输出分词结果。

自适应分词步骤如图 5.10 所示。

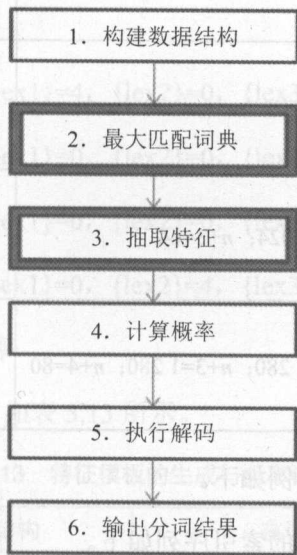


图 5.10 自适应分词步骤

其中，第 3~5 步为标准的 CRF 解码过程。下面对加粗黑框中的节点展开说明。

详细给出如下几个具体的环节的执行过程。

1) 最大匹配词典

方法主体逻辑并不复杂，就是将输入句子字符串中的每个字与其后的 n 个字试组成词($n=1,\cdots,5$)，然后与内部词典和外部词典匹配，并找出最大的匹配项。

例句：欧洲东部的罗马尼亚，首都是布加勒斯特，也是一座世界性的城市。

Ltp 系统默认的内部词典的词数并不多，仅有 8625 个词。这个步骤仅对输入的句子进行初步的切分。但是，中文中常有多字词内含更小单位词的情况。例如，“世界性”这个词中内含词“世界”；“罗马尼亚”中内含词“罗马”，等等。这就不仅需要记录最大匹配的词，还要记录内含词的下标。LTP 分词为了提高效率，下标的计算使用了位运算。

这种方式有助于提高代码的运行效率。现将位运算的结果总结出可读性较强的匹配规则，便于读者进一步理解。

不同长度词的下标列表如表 5.12 所示。

表 5.12 不同长度词的下标列表

词 长	多字词的词首字下标	算法公式
1	$n=0$	独字词为0
2	$n=2; n+1=32$	首字下标为最大匹配的词长 尾字下标为32
3	$n=3; n+1=768; n+2=48$	首字下标为最大匹配的词长 中间词下标为768 尾字下标为32+16
4	$n=4; n+1=1\ 024; n+2=1024; n+3=64$	首字下标为最大匹配的词长 中间词下标为1 024 尾字下标为32+16+16
5	$n=5; n+1=1\ 280; n+2=1\ 280; n+3=1\ 280; n+4=80$	首字下标为最大匹配的词长 中间词下标为1280 尾字下标为32+16+16+16

根据表 5.12，内含词情况举例如下。

首都：这是一个二字词。它的索引序列如下。

首	都
2	32

南斯拉夫：这是一个四字专有名词，是国家的名称。其中任何两个字都不为词。它的索引序列如下。

南	斯	拉	夫
4	1 024	1 024	64

世界性：这是一个三字词。但查询到“世界”为二字词。它的索引序列如下。

世	界	性
2	32	0
3	768	48
3	800	48

罗马尼亚：这是一个四字专有名词，也是国家的名称。但查询到“罗马”为二字词。

它的索引序列如下。

罗	马	尼	亚
2	32	0	0
4	1 024	1 024	64
4	1 056	1 024	64

其他情况依此类推。

当下标计算完成之后，这个值会转换为内部词典的特征值。以“罗马尼亚”为例，转换结果如下。

“罗”字的内部词典特征，{lex1}=4，{lex2}=0，{lex3}=0。

“马”字的内部词典特征，{lex1}=0，{lex2}=0，{lex3}=4。

“尼”字的内部词典特征，{lex1}=0，{lex2}=0，{lex3}=4。

“亚”字的内部词典特征，{lex1}=0，{lex2}=4，{lex3}=0。

2) 生成特征模板及计算概率

特征模板的生成与概率计算如表 5.13 所示。

表 5.13 特征模板的生成与概率计算

在抽取特征阶段产生的数据结构	在计算概率阶段得到的参数
0x1885040 1=_bos_ // {c-2}	key:1=_bos_ dict id:0 indx:0
0x1870e40 2=_bos_ // {c-1}	key:2=_bos_ dict id:1 indx:23152
0x1870e80 3=欧 // {c-0} 当前字	key:3=欧 dict id:2 indx:51216
0x1884310 4=洲 // {c+1}	key:4=洲 dict id:3 indx:76644
0x1884350 5=东 // {c+2}	key:5=东 dict id:4 indx:98980
0x1883ba0 6=_bos_-_bos_ // {c-2}-{c-1}	key:6=_bos_-_bos_ dict id:5 indx:120848
0x1883bd0 7=_bos_-欧 // {c-1}-{c-0}	key:7=_bos_-欧 dict id:6 indx:1866956
0x1883c10 8=欧-洲 // {c-0}-{c+1}	key:8=欧-洲 dict id:7 indx:3360008
0x188b240 9=洲-东 // {c+1}-{c+2}	key:9=洲-东 dict id:8 indx:-1
0x188b280 14=_bot_ // {c-1} chartype	key:14=_bot_ dict id:9 indx:6473268
0x188b2c0 15=0 // {c} chartype	key:15=0 dict id:10 indx:6473296
0x188b300 16=0 // {c+1} chartype	key:16=0 dict id:11 indx:6473320
0x188b340 17=2 // 二字词的词首	key:17=2 dict id:12 indx:6473348
0x188b380 18=0 // 词尾	key:18=0 dict id:13 indx:6473368
0x188b3c0 19=0 // 词中	key:19=0 dict id:14 indx:6473388

续表

在抽取特征阶段产生的数据结构	在计算概率阶段得到的参数
注意，0x188b340 17=2 是内部词典使用的特征模板。这里的2表示二字词的词首字	字段“key”是原子字符；“dic id”是特征词典序号；indx是对应着特征词典中查到的索引。例如， 0x162bafc ("3=欧", 1074687717, 12804, 1, -1) 12 804×4 = 51 216 该值可以直接在Param表中查询，得到最终的概率值

表 5.12 中，需要注意的是：key:17=2 dict id:12 indx:6473348 对应着查询到的概率。系统通过调整这个概率值来使最终的解码算法将内部词典查询到的结果从所有候选词中分离出来，作为最终的分词结果。

受篇幅所限，有关细节的概率计算等相关内容，建议读者查询 Ltp 3.2 中文分词源码解析文件，该文件位于 <http://www.threedweb.cn/thread-1396-1-1.html> 中，并通过调试代码得到结果进行验证。这里不再赘述。

5.3.4 算法的策略——层叠式架构

NLP 中常用的层叠式架构包括：层叠式 HMM 和层叠式 CRF。其中，层叠式 HMM 主要用于解决组织机构名识别问题。本节主要介绍层叠式 HMM 的算法思想和实现代码。

基于层叠式隐马尔科夫模型（Cascaded HMM, Cascaded Hidden Markov Model）的方法也是由张华平博士等提出的。其算法思想是：首先在词语粗切分的结果集上，采用底层隐马尔科夫模型识别出普通无嵌套的人名、地名和机构名等，然后依次采取高层隐马尔科夫模型识别出嵌套了人名、地名的复杂地名和组织机构名。中国科学院计算技术研究所研制的汉语词法分析系统 ICTCLAS 采用的就是基于层叠式隐马尔科夫模型的命名实体识别，该系统在第一届中文分词大赛中名列前茅。

在层叠式 HMM 的基础上，他们又提出了基于角色标注的命名实体识别方法。命名实体识别中最难的部分当属实体机构名，这是因为机构名的组成成分十分复杂，可以是人名、地名、序数词、企业字号甚至上级机构名。为了充分利用机构名构成上的特点，根据每个字词在机构名构成中的不同作用，把它们分成各个不同的角色。经过对角色集选取的反复试验，对不同的命名实体制订出不同角色表，然后再根据角色表来进行识别。

回顾第 2 章的 HanLP 中文分词，为了保证分词算法的完整性，简要介绍过人名识别

的算法部分, HanLP 的命名实体识别部分包括: 人名识别、地名识别和组织机构名识别, 从整体上就是层叠式 HMM 的具体实现。下面给出整个命名实体识别的总流程代码, 该部分代码位于 HanLP 1.28 的 `com.hankcs.hanlp.seg.NShort` 类中。

```

.....
if (config.ner) // 命名实体识别
{
    wordNetOptimum.addAll(vertexList);
    int preSize = wordNetOptimum.size();
    if (config.nameRecognize) // 中国人名识别 {
        PersonRecognition.Recognition(vertexList, wordNetOptimum, wordNetAll);
    }
    if (config.translatedNameRecognize) // 译名识别 {
        TranslatedPersonRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
    }
    if (config.japaneseNameRecognize) // 日本人名识别 {
        JapanesePersonRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
    }
    if (config.placeRecognize) // 地名识别 {
        PlaceRecognition.Recognition(vertexList, wordNetOptimum, wordNetAll);
    }
    if (config.organizationRecognize) // 组织机构名识别 {
        vertexList = Dijkstra.compute(GenerateBiGraph(wordNetOptimum));
        wordNetOptimum.addAll(vertexList);
        OrganizationRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
    }
    if (!NERexists && preSize != wordNetOptimum.size())
    {
        NERexists = true;
    }
}
}
.....

```

本节以组织机构识别实现为例, 给出命名实体识别的算法流程。该资源主要来源于 HanLP 1.28 项目的源码, 限于篇幅原因, 具体实现的部分我们做了简省。

1. 定义角色词典

现给出组织机构部分的角色表 (见表 5.14)。该表位于 `com.hankcs.hanlp.corpus.tag`

包内，是一个枚举类型，名为 enum NT。

表 5.14 HanLP角色表（来自enum NT）

角 色	意 义	例 子
A	上文	[参与]亚太经合组织的活动
B	下文	中央电视台[报道]
X	连接词	北京电视台[和]天津电视台
C	特征词的一般性前缀	北京[电影]学院
F	特征词的译名性前缀	美国[摩托罗拉]公司
G	特征词的地名性前缀	交通银行[北京]分行
H	特征词的机构名前缀	[中共中央]顾问委员会
I	特征词的特殊性前缀	[中央]电视台
J	特征词的简称性前缀	[巴]政府
K	整个机构	[麦当劳]
L	方位词	—
M	数词	公交集团[五]分公司
P	单字碎片	—
W	符号	—
D	机构名的特征词	国务院侨务[办公室]
S	句子的开头	—
Z	非机构名成分	—

生成角色词典。

下面给出从熟语料生成角色词典的过程，熟语料选自 2014 年的人民日报。以如下句子为例。

参与/v [北京/ns 电影/n 学院/nis]/nt 和/cc [美国/nsf 辛普森/nr 公司/nis]/nt 的/udel
活动/vn ， /w 由/p [交通/n 银行/nis 北京/ns 分行/n]/nt 与/cc 麦当劳/nt 赞助/v ， /w [巴
/b 政府/nis]/nt 和/cc [中共中央/nt 顾问/nnt 委员会/nis]/nt 指导/vn， /w [中央/n 电视台
/nis]/nt 报道/v

经过自动角色转换程序进行转化，部分转换步骤如下。

1) 添加句首、句尾

[始##始/S, 参与/v, [北京/ns 电影/n 学院/nis]/nt, 和/cc, [美国/nsf 辛普森/nr 公司
/nis]/nt, 的/udel, 活动/vn, , /w, 由/p, [交通/n 银行/nis 北京/ns 分行/n]/nt, 与/cc, 麦当

劳/nt, 赞助/v, , /w, [巴/b 政府/nis]/nt, 和/cc, [中共中央/nt 顾问/nnt 委员会/nis]/nt, 指导/vn, , /w, [中央/n 电视台/nis]/nt, 报道/v, 未##末/Z]

2) 标注上文

[始##始/S, 参与/A, [北京/ns 电影/n 学院/nis]/nt, 和/A, [美国/nsf 辛普森/nr 公司/nis]/nt, 的/udel, 活动/vn, , /w, 由/A, [交通/n 银行/nis 北京/ns 分行/n]/nt, 与/A, 麦当劳/nt, 赞助/v, , /A, [巴/b 政府/nis]/nt, 和/A, [中共中央/nt 顾问/nnt 委员会/nis]/nt, 指导/vn, , /A, [中央/n 电视台/nis]/nt, 报道/v, 未##末/Z]

3) 标注下文

[始##始/S, 参与/A, [北京/ns 电影/n 学院/nis]/nt, 和/B, [美国/nsf 辛普森/nr 公司/nis]/nt, 的/B, 活动/vn, , /w, 由/A, [交通/n 银行/nis 北京/ns 分行/n]/nt, 与/B, 麦当劳/nt, 赞助/B, , /A, [巴/b 政府/nis]/nt, 和/B, [中共中央/nt 顾问/nnt 委员会/nis]/nt, 指导/B, , /A, [中央/n 电视台/nis]/nt, 报道/B, 未##末/Z]

4) 标注中间

[始##始/S, 参与/A, [北京/ns 电影/n 学院/nis]/nt, 和/X, [美国/nsf 辛普森/nr 公司/nis]/nt, 的/B, 活动/vn, , /w, 由/A, [交通/n 银行/nis 北京/ns 分行/n]/nt, 与/X, 麦当劳/nt, 赞助/B, , /A, [巴/b 政府/nis]/nt, 和/X, [中共中央/nt 顾问/nnt 委员会/nis]/nt, 指导/B, , /A, [中央/n 电视台/nis]/nt, 报道/B, 未##末/Z]

5) 处理整个句子

[始##始/S, 参与/A, 未##地/G, 电影/C, 学院/D, 和/X, 未##地/G, 未##人/F, 公司/D, 的/B, 活动/Z, , /Z, 由/A, 交通/C, 银行/D, 未##地/G, 分行/D, 与/X, 麦当劳/K, 赞助/B, , /A, 巴/J, 政府/D, 和/X, 未##团/K, 顾问/C, 委员会/D, 指导/B, , /A, 中央/C, 电视台/D, 报道/B, 未##末/Z]

注: 所有标注内容均以黑体斜体字列出。该实例参考了《层叠 HMM-Viterbi 角色标注模型下的机构名识别》一文。

在对所有熟语料句子执行自动标注后, 即可统计每一个非 Z 词语的各角色词频, 最后得到一个角色词典, 如图 5.11 所示。该词典位于 Hanlp-1.28\data\dictionary\organization 目录下的 nt.txt 文件中。

1. 公司	D 4621 A 24 B 24
2. 公司总部	B 1
3. 公司治理	B 2
4. 公司股票	B 1
5. 公告	B 15 X 1
6. 公告栏	B 2
7. 公园	C 9
8. 公安	C 728 A 19 B 5
9. 公安分局	A 18
10. 公安厅	D 226 A 4
11. 公安处	D 183
12. 公安学	B 4
13. 公安局	D 2251 A 5
14. 公安机关	B 10
15.	

图 5.11 nt.txt 角色词典示例

2. 统计转移矩阵

转移矩阵是指从一个角色标签转移到另一个角色的频次，利用它和角色词频可以计算出 HMM 中的初始概率、转移概率和发射概率，进而完成求解。2015 年人民日报训练转移矩阵如表 5.15 所示。

表 5.15 2015 年人民日报训练转移矩阵

	A	B	C	D	F	G	I	J	K	L	M	P	S	W	X	Z
A	0	0	19 945	883	2 013	58 781	3 290	1 582	19 254	1 422	282	944	0	0	0	0
B	3 013	0	0	0	0	0	0	0	0	0	0	0	0	0	0	125 708
C	0	0	25 949	57 230	142	1 908	850	1 603	53	169	260	834	0	144	0	0
D	0	109 511	4 389	6 473	135	1 018	476	229	28	105	177	120	0	59	4 586	0
F	0	0	971	2 666	138	127	92	163	5	7	5	87	0	48	0	0
G	0	0	24 497	42 962	1 283	5 178	3 104	2 782	182	1 415	756	1 173	0	140	0	0
I	0	0	2 515	5 556	34	270	179	181	3	39	56	210	0	11	0	0
J	0	0	2 002	4 973	69	96	85	707	4	95	47	535	0	23	0	0
K	0	19 920	1 162	2 036	13	184	64	57	16	3	32	25	0	0	1 238	0
L	0	0	1 289	1 476	19	68	58	481	1	18	10	289	0	0	0	0
M	1 000	0	569	758	1	7	11	128	0	76	86	143	0	1	0	0
P	0	0	1 647	1 989	70	54	200	425	0	67	29	433	0	9	0	0

续表

	A	B	C	D	F	G	I	J	K	L	M	P	S	W	X	Z
S	9 509	0	2 911	104	278	12 704	476	212	4 031	210	23	86	0	0	0	1 131 650
W	0	0	123	150	23	105	11	9	0	4	0	10	0	0	0	0
X	0	0	1 173	50	91	2 972	158	77	1 173	79	17	34	0	0	0	0
Z	95 874	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19 796 133

如表 5.14 所示，该矩阵的文本文件位于 Hanlp-1.28\data\dictionary\organization 目录下的 nt.tr.txt 文件中。

3. 识别算法

HanLP 的命名实体识别代码位于 com.hankcs.hanlp.seg.NShort 类中，如下节选的代码为组织机构识别部分。

```
if (config.organizationRecognize)
{
    vertexList = Dijkstra.compute(GenerateBiGraph(wordNetOptimum));
    wordNetOptimum.addAll(vertexList);
    OrganizationRecognition.Recognition(vertexList, wordNetOptimum,
wordNetAll);
}
```

因篇幅所限，我们就不给出算法的源码解析过程。如下是《层叠 HMM-Viterbi 角色标注模型下的机构名识别》一文中的一个例子，对原文内容我们做了修改，便于读者领略整个算法的过程。

济南杨铭宇餐饮管理有限公司是由杨先生创办的餐饮企业

在组织机构名识别前，会得出如下输出。

[济南/ns, 杨铭宇/nr, 餐饮/n, 管理/vn, 有限公司/nis, 是/vshi, 由/p, 杨先生/nr, 创办/v, 的/ude1, 餐饮/n, 企业/n]

上例中该公司的各个成分被拆散，无法构成完整的机构名。根据分词结果进行组织机构名的角色标注。

组织机构名角色观察如下。

[S 1162194][济南 G 83472 B 1200 A 470 D 84 X 4][杨铭宇 F 4309 B 769 A 266 D 254 X 6][餐饮 C 58 B 12][管理 C 706 B 70 A 5][有限公司 D 2861 A 1 B 1][是 A 2340 B 353 X 20 P 2][由 A 1579 B 16 X 11][杨先生 F 4309 B 769 A 266 D 254 X 6][创办 A 20 B 5][的

B 7092 A 4185 X 20][餐饮 C 58 B 12][企业 C 86 A 42 B 11 X 6][B 710]

组织机构名角色标注如下。

[/S,济南/G,杨铭宇/F,餐饮/C,管理/C,有限公司/D,是/B,由/A,杨先生/F,创办/A,的/B,餐饮/C,企业/C,/B]

模式匹配如下。

系统提供了各类组织机构名称的角色模式串库。当上文识别出角色标注串与组织机构的模式串匹配时，就认为识别出了一个组织机构。这些模式串由组织机构的角色表中的每个角色构成，保存在 com.hankcs.hanlp.dictionary.nt 下的 OrganizationDictionary 类中。这些模式共有 3 661 个，数量众多，无法在这里全部列出。我们已经将其整理为一个 txt 模式文件，有兴趣的读者可以从 <http://www.threedweb.cn/thread-1590-1-1.html> 下载。

HanLP 使用 Aho-Corasick 算法来进行模式匹配，Aho-Corasic 是一个基于确定有限状态自动机快速模式匹配算法，算法内容详见 <http://www.hankcs.com/program/algorithm/implementation-and-analysis-of-aho-corasick-algorithm-in-java.html>。因与本节主要内容关系不大，有兴趣的读者可以参看原文。匹配之后模式串如下。

```
CCCD
PPD
PPDCD
PPFCCD
PPFCD
.....
```

经过细分，识别出机构名：济南杨铭宇餐饮管理有限公司 GFCCD。例句的命名实体输出如图 5.12 所示。

```
to: 8, from: 7, weight:04.65, word:是@由
to: 9, from: 8, weight:02.02, word:由@未##人
to: 10, from: 9, weight:09.11, word:未##人@创办
to: 11, from: 10, weight:01.14, word:创办@的
to: 12, from: 11, weight:06.41, word:的@餐饮
to: 13, from: 12, weight:01.82, word:餐饮@企业
to: 14, from: 13, weight:06.20, word:企业@未##未

[济南杨铭宇餐饮管理有限公司/nt, 是/vshi, 由/p, 杨先生/nr, 创办/v, 的/udel, 餐饮/n, 企业/n]
```

图 5.12 例句的命名实体输出

需要说明的是，这仅仅是层叠式隐马尔科夫模型的最后一层。整个组织机构的识别

经历了人名识别（中国人名、日本人名、其他译名）、地名识别最终完成到组织机构名称的识别。这个漫长的流程得出的最终结果如下。

[济南杨铭宇餐饮管理有限公司/nt, 是/vshi, 由/p, 杨先生/nr, 创办/v, 的/ude1, 餐饮/n, 企业/n]

5.4 结语

本章的主要内容是基于概率图模型的算法原理，结合相关开源框架，实现的 NLP 序列标注的基本任务的示例章节。全部章节分为如下三个部分。使用最大熵模型实现汉语词性标注；使用条件随机场模型实现语义组块标注；最后使用自适应的条件随机场模型和层叠式 HMM 实现命名实体识别。

每一节都可以划分为如下两大部分：语言知识部分和算法实现部分。在语言知识部分中，尽可能多地给出汉语中不同的句法和语义的规则，有的甚至给出了多种规则。例如，在词性标注章节，给出了两种汉语词性的标注规范，并对两种规范做了比较。差异的部分，都一一做了说明，并提供了比较列表。在语义组块章节，对比较重要的 NP、VP 进行了全面的分析，对于其他语义组块，也做了说明。

最复杂的是命名实体识别。首先介绍了命名实体识别的主要类别和困难，并举例说明。在此基础之上，给出了两种命名实体识别的分词架构，并对每种架构的处理逻辑都给出了说明。在算法部分，首先，详细介绍了自适应 CRF 算法的关键过程。之后，详细介绍了层叠式 HMM 算法的从模式抽取到语料训练，直至实体识别的详细过程。读者可以对照源码进一步学习，以加深理解。

受篇幅所限，对于每节在语言知识方面的内容，我们都做了最大限度的简化，只保留了最重要的模式，而忽略了论述部分。如果读者想要进一步研究，需要参考章节中给出的相关文档。

第 6 章

句法理论与自动分析

印欧语系与汉语的诸多不同之中，有如下两点是至关重要的。第一，印欧语系都实行分词连写，词与词之间用空格分割，因此没有分词的问题。第二，印欧语种大多数都通过形态变化构造语法结构，有很强的规范性。基于印欧语系这些特点，在自然语言诞生的初期阶段，句法分析已经成为西方 NLP 的核心主题。这是内因。

最早的自然语言处理的研究工作就是机器翻译，它部分来源于政治需要。1949 年，美国人威弗首先提出了机器翻译设计方案，直到 20 世纪 60 年代末进入低潮前，机器翻译都是 NLP 应用的主要领域，并且耗费了巨额费用。迄今为止，机器翻译的研究虽然取得了很大的进展，但是仍旧没有达到实用的程度。而句法分析又是机器翻译最核心的数据结构，因此多年来，句法研究是国际自然语言处理的重中之重。这是外因。

句子的分析，也被称作对语言进行深层的处理，主要技术包括句法分析和语义解析。它们都需要对句子进行全局分析，而句法分析曾经一直处于核心的位置。目前，句法的语言分析技术还没有达到完全实用的程度。直到现在，主要语种中都没有一个能够达到工业精度的句法解析器。

本章所讲的句法分析理论是多年来 NLP 领域应用最为广泛的两种，分别为转换生成语法和依存句法。其中，转换生成语法规则由美国语言学家乔姆斯基在 20 世纪 50 年代创立，历史悠久，著名的宾州树库就使用这种句法规则进行手工标注。而依存句法是目前实际应用最多的一种句法理论。较之转换生成句法，该种理论相对简单，可以较大地提高识别精度，现在几种比较新的句法解析器都支持以依存树库作为训练资源。值得庆

幸的是,转换生成句法标注的句子可以自动转换为依存句法树的形式,资源的共用极大地减少了重复投入。

随着人们对语言研究的逐渐深入,人们逐渐发现语义的重要性。近年来,语言学家达成了共同的观点:“凡是由实词和虚词构成的汉语句子里面,一定同时并存着两种结构关系,它们分别是句法关系和语义关系。而这两种结构关系间并不是一一对应的。”(陆俭明)。其中,句子的语义结构是语言的意义的表达方式,句子的语法结构是语言信息传递的编码结构。句子同时受到语义和语法的共同影响,才形成了现实中千差万别的句子。这个发现对于缺乏形态变化的汉语而言尤其重要。那么,以往以构建句法树作为语言处理的终极任务,现在变为正确地识别出句子中的各种句法成分。

在通向高精度语义识别的各种道路中,虽然句法解析被赋予了新的定位和角色,但依旧是语义识别的一个重要的组成部分。

6.1 转换生成语法

诺姆·乔姆斯基(Noam Chomsky, 1928—),当代最著名的语言学家和语言哲学家,创立了转换生成语法理论。这一理论不仅获得了语言学界很高的评价,而且在心理学、哲学、逻辑学等方面引起了人们普遍的重视。1972年诺姆·乔姆斯基当选为国家科学院院士,1984年获美国心理学会颁发的杰出科学贡献奖。

6.1.1 乔姆斯基的语言观

乔姆斯基的伟大之处在于他重新确立了人们的“语言”观。

“语言”在生成语法学中被界定为如下。

句子的(有限的或无限的)集(set),每个句子在长度上是有限的,它由结构成分有限的集构成。
——(Chomsky 1957,《句法结构》)

自然语言中的每一个具体的语种都是无限的句子集合,是在有限的规则基础上生成无限多的句子的集合。其特征如下。

(1) 无限性:自然语言是一个无限集,其中的元素是句子。语言是以有限的手段作无限的运用。

(2) 离散性：以有限的符号构成无限的符号序列；任何连续的话语都可以切分为更小的片段。

(3) 结构层次性：线性特征中蕴涵着层次关系。这一特征是描写主义语言学揭示出来的。

“语法是研究具体语言里用以构造句子的原则和加工过程。”(Chomsky 1957)。一种语言的语法是一种装置，它不仅应该能生成该语言中所有的句子，而且只能生成合格的(Well-Formed)句子。具体语言的语法表现为一套有限的规则系统。这个系统由如下三个子系统构成：句法部分、音系部分和语义部分。

关于语法和语言的关系，转换生成语法给出了如下基本假设。

- ❑ 天赋观：人们的语言能力是先天就有的，核心语法人无须学习，而其他边缘部分，需要后天学习。
- ❑ 普遍观：人的语法知识包括两个部分。一部分是全人类特有的，称为普遍语法(Universal Grammar)；另一部分是具体语种所具有的，称为“个别语法”(Particular Grammar)。普遍语法表现为一套有限的原则和数量极少的参数。
- ❑ 自治观：人们头脑中生来就有专司语言加工的独立机制，句法不用参照意义和其他因素就能对语句从形式上作出系统描述。
- ❑ 模块观：乔姆斯基假设出一套模块论分析方案，将语言分成音位子模块、句子模块和语义子模块。
- ❑ 二元论与形式观：心智中的句法操作可以完全独立于意义而存在，语言中的句法可运用一套纯形式的句法公式来加以演算。因此，某一种语言中的全部合乎语法的句子就是基于一套形式化的符号、通过一定的规则对其进行形式操作而生成出来的。

——《构式语法》

因此，乔姆斯基认为语法是第一性的，而具体的语言则是派生性的，是由在语法基础上构成的无限多的具体句子所构成的集合。

语法可以看作一种“内在性语言”(I-Language)，具体的语言则是一种“外在化语言”(E-language)。转换生成语法的研究目标不是描述语言的表层结构，而是揭示语言生成的内在机制。在从某种意义上说，所有的语法都是人类在交流中由大脑生成的，那么语法规则的最终目的就是描写生成句子的这些思维机制。而且，语法规则都是从几条普遍原则转换而来的，因此称为转换生成语法。转换的步骤是有限的，而过程是递归的。

乔姆斯基于1957年后做了几年研究,到1965年便建立起一个完整的生成语法系统,包括语类、转换、音系和语义4个子系统,各子系统之间有一定的顺序关系。每个子系统都有一套规则,规则之间有一定的使用顺序,像用数学公式一样,逐步推导出句子来,不同的规则推导出不同的句子。这样,生成语法系统好比一部机械装置,运转起来能够生成某种语言中的一切合格的句子,而且只能生成那些合格的句子。当然,毋庸置疑,系统在构造和实践中,也出现了很多新的问题,以生成语法系统为基础,乔姆斯基的理论形成了如下5个不同的历史阶段。

- 古典理论阶段。
- 标准理论时期。
- 扩充标注理论时期。
- 管辖和约束理论时期。
- 最简方案时期。

这些各个历史阶段的具体内容,这里不再赘述,有兴趣的读者可以参考乔姆斯基的相关著作。下面章节着重谈论一下,乔姆斯基的转换生成语法对句法解析的一些重要贡献。

6.1.2 短语结构文法

在第三章和第四章的内容中,我们主要做了两件事,将句子的中每个字依照词典和模型归并成词汇,再将这些词汇按照词性和上下文等特征归并为短语(语义组块)。综合来看为两个步骤,其实就是自底向上逐层归纳的过程。

在分析一个句子时,也使用这种思路。将句子中的句法构造层次考虑进来,并按其构造层次,逐层进行分析,在分析时,指出每一层面的直接组成成分:词汇、短语、小句和句子。这种分析方法就是乔姆斯基对语言学最大的贡献,即短语结构文法。

1. 短语结构文法

短语结构文法可以形式化定义为 $G=(X,V,S,R)$ 这样一个四元组。

X 是一个有限词汇的集合(词典),它的元素称为词汇或终结符。

V 是一个有限标注的集合,叫作非终结符集合,它的元素称为变量或非终结符集合。

$S \in V$, 称为文法的开始符号。

R 是有序偶对 (α, β) 的集合。 α 是集合 $(X \cup V)$ 上的字符串,但至少包含一个非终结符;

β 是集合 $(X \cup V)^*$ (这里 $*$ 表示闭包) 的元素。一般的, 将有序偶对 (α, β) 记为 $\alpha \rightarrow \beta$, 称为产生式。这里 “ \rightarrow ” 读作 “定义为”。

注意: 第一个产生式的左边只能有一个符号, 就是开始符号 S 。

在自然语言处理中, 终结符均为词汇、数字和标点符号。非终结符可以是词性、短语标注、小句标注等符号。

推导的定义: 给定文法 G , α 和 β 是集合 $(X \cup V)$ 上的字符串, 若 α 、 β 可以分别写成 pvr 和 pur (p 和 r 可能同时为空串), 而 $v \rightarrow u$ 是文法 G 的一个产生式, 则称串 α 可以直接推导出串 β 。记为: $\alpha \Rightarrow \beta$ 或 $pvr \Rightarrow pur$ 。

实际上所谓 “推导”, 就是用右侧的串中的 u 替代左侧的串中的 v 。

除标准的推导定义之外, 同时还定义了如下两个推导符号。

□ 用符号 $y \Rightarrow^* z$ 表示 y 可以经过任意步 (包括 0 步) 推导出 z , 即

➤ $y = z$ 。

➤ 存在一个序列: $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$, 有 $y = \alpha_1$, $z = \alpha_n$, 且 $\alpha_i \Rightarrow \alpha_{i+1}$, 对所有 $i \geq 1$ 。

□ 用符号 $y \Rightarrow^+ z$ 表示 y 可以经过多步 (最少 1 步) 推导出 z , 即

存在一个序列: $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$, 有 $y = \alpha_1$, $z = \alpha_n$, 且 $\alpha_i \Rightarrow \alpha_{i+1}$, 对所有 $i \geq 1$ 。

下面给出句型 and 句子的定义。

对于文法 G , 如果 $S \Rightarrow^* w$, 则称 w 是文法 G 的一个句型, 若 w 中包含的字符全是非终结符, 则称 w 是一个句子。

有了句型和句子的定义, 自然可以得到语言的定义。所谓语言, 在乔姆斯基的短语结构语法中定义如下。

给定文法 G , 有开始符号 S , 则把 S 可以推导出的所有的终结字符串的集合 (所有的句子集合) 称为由文法产生的语言, 记为 $L(G)$ 。

注意: 对于文法和语言, 可以从一个文法得到该文法产生的语言; 也可以根据语言, 写出产生该语言的某种文法。在后面讲的 PCFG 等算法中, 从句子产生文法的过程称为训练或学习过程, 从文法产生语言的过程称为生成过程。

2. 上下文无关文法

乔姆斯基的短语结构文法分为上下文无关文法和上下文有关文法两类。因为两者从

本质上差别不大,绝大多数上下文有关文法产生的规则都可以转换为上下文无关文法。这里主要研究上下文无关文法的概念。

定义:给定文法 G , 如果对于 G 中的任意产生式 $v \rightarrow w$, 而 v 只是一个非终结符, 即 $A \rightarrow w$, $A \in V$, $w \in (X \cup V)^*$, 则称文法 G 为上下文无关文法 CFG。

下面看一个简单的实例, 假设有一个上下文无关文法的产生式。

(1) $S \rightarrow NP, VP$. (5) $NN \rightarrow [\text{会议}]$.

(2) $VP \rightarrow VP, AS, NP$. (6) $NR \rightarrow [\text{张三}]$.

(3) $NP \rightarrow NN$. (7) $VV \rightarrow [\text{参加}]$.

(4) $NP \rightarrow NR$. (8) $AS \rightarrow [\text{了}]$.

另外有一个例句, 该句是经过分词、词性标注之后得到的结果, 也可以看作事先应用了第(5)~(8)条规则。

张三/NR 参加/VV 了/AS 会议/NN

我们希望对上述例句使用上下文无关文法推导出其层次结构。

推导过程如下。

首先换一下形式, 将例句换为宾州树库的表示方法。

(NR 张三) (VV 参加) (AS 了) (NN 会议)

按照上述文法产生式(规则), 分别应用第(3)条和第(4)条规则。这样就得到如下两个 NP。

(NP (NR 张三)) (VV 参加) (AS 了) (NP (NN 会议))

再次应用上述文法规则中的第(2)条, 也就是说 VP 要与跟在它后面的体词 AS 和它的宾语 NP 结合到一起, 形成一个完整的 VP, 于是上式变为如下内容。

(NP (NR 张三)) (VP (VV 参加) (AS 了) (NP (NN 会议)))

最后应用第(1)条规则, 设置整个句子的根节点为 S (如果自动标注, 会选择简单句的标签 IP, 或 ROOT 表示为整个句子的根节点), 最终结果就写成如下内容。

(S (NP (NR 张三)) (VP (VV 参加) (AS 了) (NP (NN 会议))))

这就是我们推导的最终结果, 这个结果称为推导树, 也称为句法树的数据表示。推

导树除具有数据的表示形式还有图形的表示形式，其图形表示就是一棵我们常见的句法树。

推导树（句法树）可以表示为一棵有向无环图。树的根节点是文法的开始符号 S （当然也可以是 $ROOT$ ）。树的其他节点是文法中的非终结符或终结符。

如果推导使用了产生式 $A \rightarrow \alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$ ，其中， A 是非终结符， α_i 可以是终结符或非终结符，则 $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$ 都是 A 的直接后继节点，其中 A 称为父节点， $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$ 称为子节点。若节点是终结符，则该节点称为叶子节点。若节点是非终结符，则该节点称为非叶子节点，或直接简称节点。

需要说明的是，推导树不仅可以表示句子的产生，也可以表示一个句型，但句型的叶子节点都是非终结节点。

在推导树中，从根节点到一个叶子节点的边的集合称为一条路径，一条路径上的非终结符的个数称为路径的长度。最大的路径长度就是该推导树的高度。

在一棵推导树 T 中，以任意一个非终结符 A 为根，连同它的所有后继节点（直接的节点和非直接的节点），构成一棵子树，称之为推导树 T 的 A -子树。本书中推导树就是树 T 的最大的一棵子树，即 S -子树。

在推导树（句法树）中，一个节点对其所有下位节点构成“支配”（Dominate）关系。支配是一种由上到下的纵向关系。在有支配关系的两个节点中，若没有其他节点出现，这样的支配关系称为“直接支配”（Immediately Dominate）关系。在直接支配关系中，处于支配地位的节点称为父节点。同属一个父节点的两个或两个以上的节点称为兄弟节点。

最后，使用一个 NLTK 库的程序来显示这棵上述例子中推导树的树状图，如图 6.1 所示。

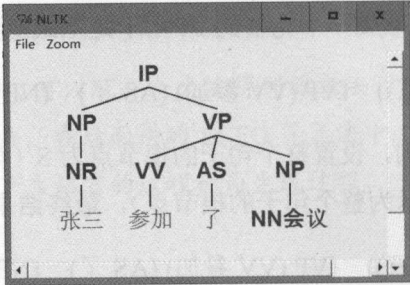


图 6.1 短语结构推导树

最终生成的推导树，也就是我们常说的句法树，就好像一棵倒置的大树。其中，“IP”称为“根”，一般用“ROOT”来表示，“根”下面每个非终结符出现的位置就是节点，出现在终结符位置的词语串就是叶子节点，连接根和节点、节点和节点、节点和叶子节点之间的线称为“边”，边所代表的含义就是上述8条产生式规则集。其中，根IP有两个分枝，称为二叉分枝（说明产生式的右侧有两项）。一个节点如有三个分枝则称为三叉分枝（说明产生式的右侧有三项）。如图6.1中的VP就分叉为VV、AS、NP三个枝。

图6.1中，ROOT节点不仅支配VP节点和第一个NP节点，也支配其下不同层次上的其他节点。VP及其下位节点为ROOT所支配，但VP只能支配自己的下位节点。直属于ROOT的NP节点或说受ROOT直接支配的NP节点是句子的主语，直属于VP的NP节点或说受VP直接支配的NP节点是句子的宾语。

上下文无关文法的直接结果有如下两个。

- (1) 句法树数据表示：使用“()”或“[]”作为成分切分符号的嵌套格式的句法树。
- (2) 句法树图形表示：短语结构图。

下面我们给出实现代码。

NLTK的程序代码也很简单，内容如下。

```
# -*- coding: utf-8 -*-

import sys, os
import nltk
from nltk.corpus import treebank
from nltk.tree import Tree

reload(sys)
sys.setdefaultencoding('utf-8')

sentTree = "(IP (NP (NR 张三)) (VP (VV 参加) (AS 了) (NP (NN 会议)))))"

tree = Tree.fromstring(sentTree)
tree.draw()
```

同时，还给出 stanford PCFG Parser 的 Java 运行代码，内容如下。

```
package edu.stanford.nlp.ademo;

import java.util.List;
import edu.stanford.nlp.ling.CoreLabel;
```



```

import edu.stanford.nlp.parser.lexparser.LexicalizedParser;
import edu.stanford.nlp.ling.Sentence;
import edu.stanford.nlp.trees.Tree;

public class PCFGDemo {

    public static void main(String[] args) {
        LexicalizedParser lp = LexicalizedParser
            .loadModel("models/lexparser/chinesePCFG.ser.gz");
        String[] sent = { "张三", "参加", "了", "会议" };
        List<CoreLabel> rawWords = Sentence.toCoreLabelList(sent);
        Tree parse = lp.apply(rawWords);
        parse.pennPrint();// 输出的结果类似于语法分析树
        System.out.println();
    }
}

```

执行结果如下。

```

(ROOT
  (IP
    (NP (NR 张三))
    (VP (VV 参加) (AS 了)
      (NP (NN 会议)))))

```

使用句法树来表示句子的结构不是计算语言学的专利，在语言教育等相关领域，也都用句法树来表示句子的分析结果。句法树也是人类学习语言、分析语言的助手。句法树表示一个句子的层次结构不是偶然，它是直观的句子形式模型描述，无论是人还是机器都很容易理解，便于作为一种统一的句子理解的结果表述。本节所展示的句法树形式为短语结构文法的一种上下文无关文法。这类句法树侧重于句子的语法结构，是短语结构文法研究句子结构的标准形式。

句法树，以及由句法树构成的树库是机器自动学习自然语言句法知识的重要资源。因此，句法树标注方式的好坏直接影响自动学习的效率和实现。需要指出的是，由于句法解析的精度还未达到所需的要求，而句法树又是从句子形式到句子语义识别的一个至关重要的环节，因此句法树的最终标注形式还未统一。本书提供了如下两种最常用的句法树形式：短语结构文法和依存句法。

计算语言学研究句法树的最终目的是为了使机器能够正确、全面地理解和表达句义，从而转换为知识库所需的数据结构，达到存储知识、实现推理的目标。即便句法解析的

精度达到工业级的要求, 也需要一个从句法树直接转换为知识库所需的 RDF 的阶段。由于两种数据结构的差异性, 目前转换工作仍旧处于实验阶段。

6.1.3 汉语句类

本文将汉语句子分为简单句、复合句和复句三种类型。

(1) 简单句: 全句只有一个主谓结构。在标签上, 除根节点是 IP, 其他各层的所有节点中都没有 IP 层。但不包括“把”字句、“被”字句、连动句等特殊句式。下面以句法树的形式, 给出几个简单句示例。

① 一般主谓结构 (见图 6.2)。

```
( (IP-HLN (NP-SBJ (NP-PN (NR 上海)
                        (NR 浦东))
                    (NP (NN 开发)
                        (CC 与)
                        (NN 法制)
                        (NN 建设)))
                (VP (VV 同步)))) )
```

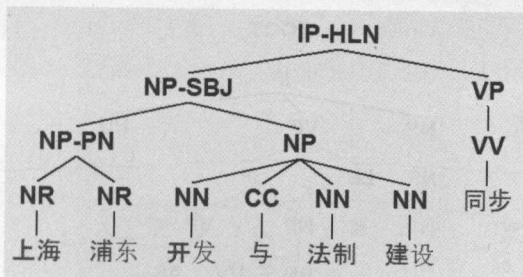


图 6.2 一般主谓结构

② “把”字结构 (见图 6.3)。

```
(ROOT (IP (NP (PN 你))
  (VP
    (ADVP (AD 简直))
    (VP (BA 把)
      (IP
        (NP (PN 我))
        (VP (VV 吓死) (AS 了))))))
  (PU 。)))
```

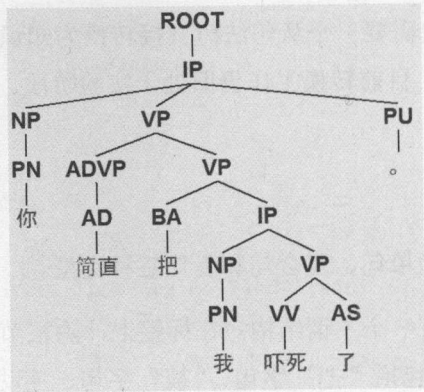


图 6.3 “把”字结构

③ “被”字结构（见图 6.4）。

```
(ROOT (IP
  (NP (NR 小鸟))
  (VP (LB 被)
    (IP
      (NP (PN 他们))
      (VP (VV 吓跑) (AS 了))))
  (PU 。)))
```

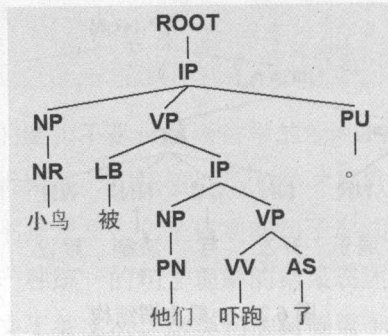


图 6.4 “被”字结构

④ 致使结构（见图 6.5）。

```
(ROOT (IP
  (NP (NN 台风))
  (VP (VV 使)
    (IP
      (NP
        (DP (DT 所有))
        (NP (NN 房子)))
      (VP
```


(ADVP (AD 都))
 (VP (VV 毁) (AS 了))))
 (PU 。)))

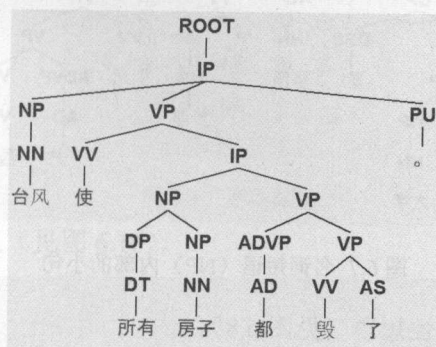


图 6.5 致使结构

(2) 复合句: 在一个单句或复杂句的某个单句子结构中, 在某类组块的内部出现 CP/IP 形式的成分, 或某个组块本身就是由 CP/IP 构成。

① 介词短语 (PP) 内部的小句 (见图 6.6)。

((IP (NP-SBJ (NN 外商) (NN 投资) (NN 企业)) (VP (PP-TMP (P 在) (LCP (IP (NP-SBJ (-NONE- *PRO*)) (VP (VV 改善) (NP-OBJ (NP-PN (NR 中国)) (NP (NN 出口) (NN 商品) (NN 结构)))))) (LC 中))) (VP (VV 发挥) (AS 了) (NP-OBJ (ADJP (JJ 显著)) (NP (NN 作用)))))) (PU 。)))

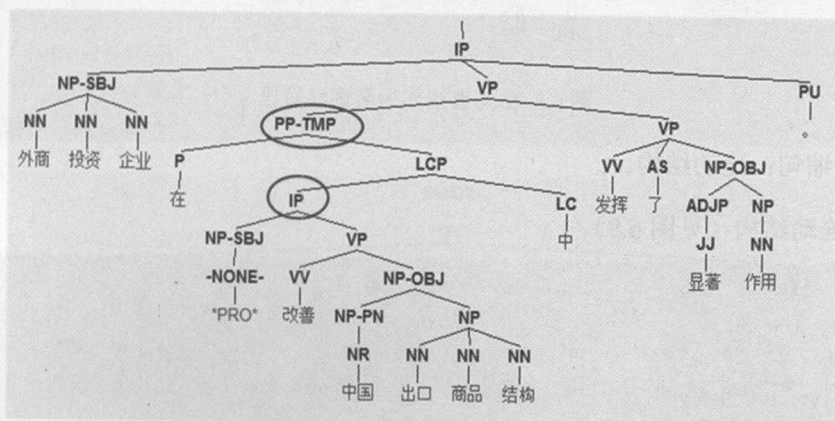


图 6.6 介词短语 (PP) 内部的小句

② 名词短语 (NP) 内部的小句 (见图 6.7)。

(ROOT (IP (NP (CP (IP (NP (PN 他)) (VP (VV 考上) (NP (NN 大学)))) (DEC 的)) (NP (NN 事情))) (VP (PP (P 在) (NP (NN 乡里))) (VP (VV 引起) (VP (ADVP (AD 一片)) (VP (VV 轰动)))))) (PU 。)))

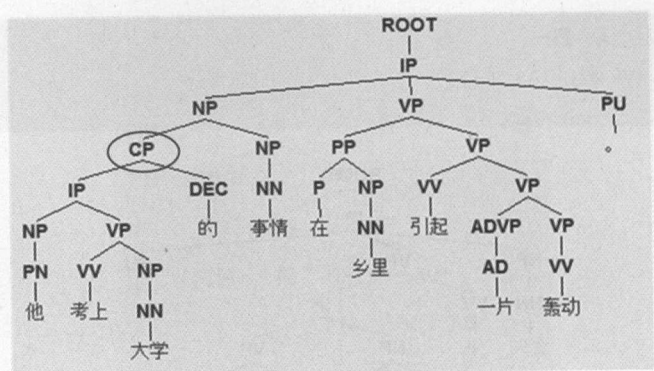


图 6.7 名词短语 (NP) 内部的小句

③ 小句直接作为名词性短语 (见图 6.8)。

(ROOT (IP (NP (CP (IP (NP (PN 他)) (VP (VV 说))) (DEC 的))) (VP (ADVP (AD 都)) (VP (VV 不对。))))))

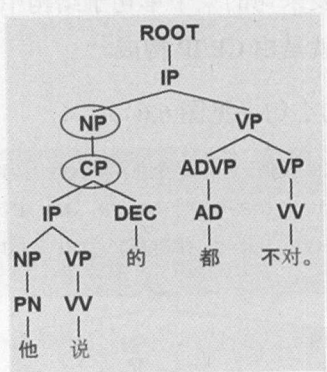


图 6.8 小句直接作为名词性短语

④ 紧缩句：连动结构。

□ 连动结构 (见图 6.9)。

(ROOT (IP (NP (PN 他)) (VP (VP (VV 上街) (VP (VV 买) (NP (NN 书)))) (VP (VV 去) (AS 了))) (PU 。)))

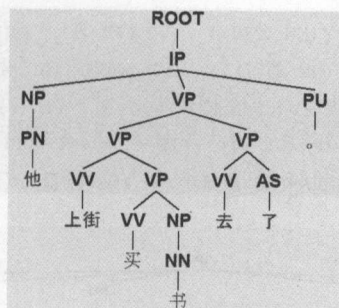


图 6.9 连动结构

□ 主语脱落的情况 (见图 6.10)。

```

(ROOT (IP
  (NP
    (NP (PN 他们))
    (NP (NN 手)))
  (VP
    (VP (VV 拉) (AS 着)
      (NP (NN 手)))
    (PU , )
    (VP (VV 穿过)
      (NP (NN 树林)))
    (PU , )
    (VP (VV 翻过)
      (NP (NN 山坡)))
    (PU , )
    (VP (VV 回到)
      (NP (NN 草房))))
  (PU 。)))
  
```

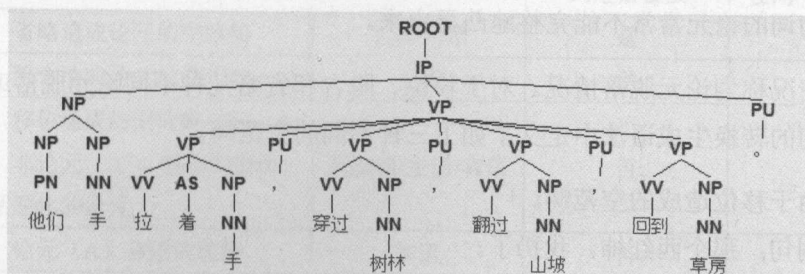


图 6.10 主语脱落的情况

(3) 复句: 全局有多个主谓结构, 是由多个简单句和简单复合单句构成的句群。

复句例句 (见图 6.11)。

((IP (IP (NP-SBJ (NN 建筑) (NN 公司)) (VP (VV 进) (NP-OBJ (NN 区)))) (PU ,) (IP (NP-SBJ (ADJP (JJ 有关)) (NP (NN 部门))) (VP (ADVP (AD 先)) (VP (VV 送上) (NP-OBJ (DP (DT 这些)) (NP (NN 法规性) (NN 文件)))))) (PU ,) (IP (NP-SBJ (-NONE- *pro*)) (VP (ADVP (AD 然后)) (VP (VE 有) (IP-OBJ (NP-SBJ (ADJP (JJ 专门)) (NP (NN 队伍))) (VP (VV 进行) (NP-OBJ (NN 监督) (NN 检查)))))) (PU 。)))

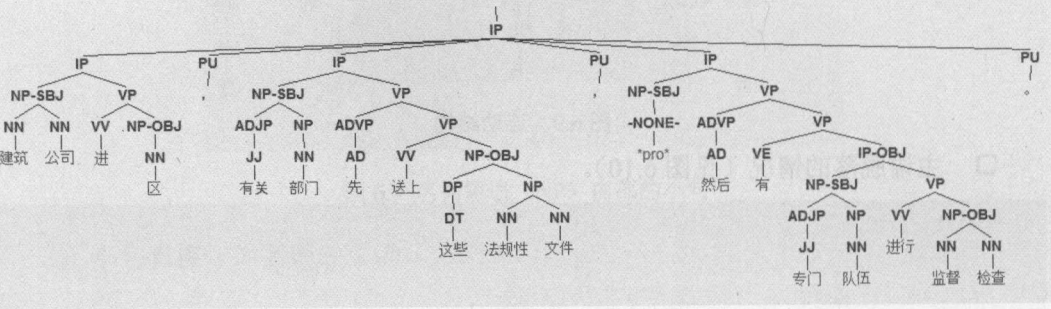


图 6.11 复句例句

从句法解析的角度看，复句由最少两个单句构成，复句中常有大量的省略成分，一般用空范畴（*pro*）标注出来。有关空范畴的详细说明请参照下文。

6.1.4 谓词论元与空范畴

配价理论认为在一个句子中，动词处于中心地位，动词的支配成分相当于传统语法的主语、直接宾语和间接宾语。配价中的价就是指动词对句子中名词性成分的支配数量，也称为论元数量，不同支配数量的动词有不同的论元（有关论元详细说明，参见第 9 章的相关解释）。一般来讲，一个动词的核心论元数量都是固定的，也就是说谓语的核心论元在句子中必须是固定的，而且凸显出来。但在实际的句子中，因为移位、隐含和省略等因素，动词的论元常常不能完整地凸显出来。

这种情况称为论元脱落情况。对于移位、隐含和省略几种不同论元脱落或移位的情况，在最初的转换生成语法中定义了如下三种不同的空范畴。

□ 由于移位造成的空范畴。

例句：那个西红柿，我扔了。

上述例句中“西红柿”被移动到了“扔”之前。该句恢复移位后为：
我扔了那个西红柿。

为了说明，转换生成语法使用 Trace 或 T 填充句子中被移位的空位置，j 表示移位词汇的位置。使用移位空范畴进行标注的结果为：

那个西红柿(j)，我扔了 T(j)。

□ 由于隐含造成的空范畴。

例句：张三打算游泳。

上述例句中“游泳”的主语与整句的主语“张三”重复，在同一句子中为了避免重复，后面的动作的主语做了隐含。但这种隐含是可以被显示恢复的。原句恢复为：

张三打算（张三）游泳。

为了说明，转换生成语法使用 PRO 标签填充句子中隐含的空位置，j 表示显性词汇的位置。使用隐含空范畴进行标注的结果为：

张三(j)打算 PRO(j)游泳。

□ 由于省略造成的空范畴。

例句：张三买了三斤苹果，给了他弟弟三个。

上述例句中，从句“给”的主语与主句的主语“张三”重复，为了避免重复，从句部分做了省略。这种省略也是可以被显示恢复的。原句恢复为：

张三买了三斤苹果，（张三）给了他弟弟三个。

为了说明，转换生成语法使用 pro 标签填充从句中省略的空位置，j 表示显性词汇的位置。使用省略空范畴进行标注的结果为：

张三(j)买了三斤苹果，pro(j)给了他弟弟三个。

表 6.1 所示为宾州树库的所有空范畴标注及其含义，并且给出了空范畴的可替换性等特征。

表 6.1 宾州树库的所有空范畴标注及其含义

	空范畴含义	空语类的位置	是否能被显性名词短语替换	先行词是否必须在同一个句子中
pro	省略造成论元的空范畴	主语/宾语	是	否
PRO	隐含造成论元的空范畴	主语	是	否*
T	移位造成的空范畴(Trace)， 非论元(A')移位的语迹， 话题化的论元	附加语/主语/宾语	否	是
*	论元(A)移位的语迹	宾语	否	是
OP	在从句结构中的操作算子。 用来指示出移位等空范畴	从句	否	否
RNR	提升从句的右部节点	主句	是	否

续表

	空范畴含义	空语类的位置	是否能被显性名词短语替换	先行词是否必须在同一个句子中
?	其他未知的空范畴	—	—	—

注意，否*：在主语控制或宾语控制的情况下，*PRO*的先行词必须处于同一个句子中。但还有第三种控制称为任意控制。在任意控制中，*PRO*的先行词可以是任意成分，不必出现在同一个句子中。

在短语结构中空范畴的表示如表 6.2 所示。

表 6.2 在短语结构中空范畴的表示

(XP (-NONE- *T*))	XP的移位语迹，诸如话题化 (Topicalization)
(NP (-NONE- *))	NP的移位语迹
(NP (-NONE- *PRO*))	隐含主语NP的
(NP (-NONE- *pro*))	脱落论元 (省略，主语/宾语)
((-NONE- *OP*))	用于关系结构的空OP算子
(WHPP (-NONE- *OP*))	用于特殊疑问句的空PP算子
(XP (-NONE- *RNR*))	用于右侧节点提升的空语类
(XP (-NONE- *?*))	省略占位符

(1) (-NONE- *T*)：A'移位语迹——话题化 (见图 6.12)。

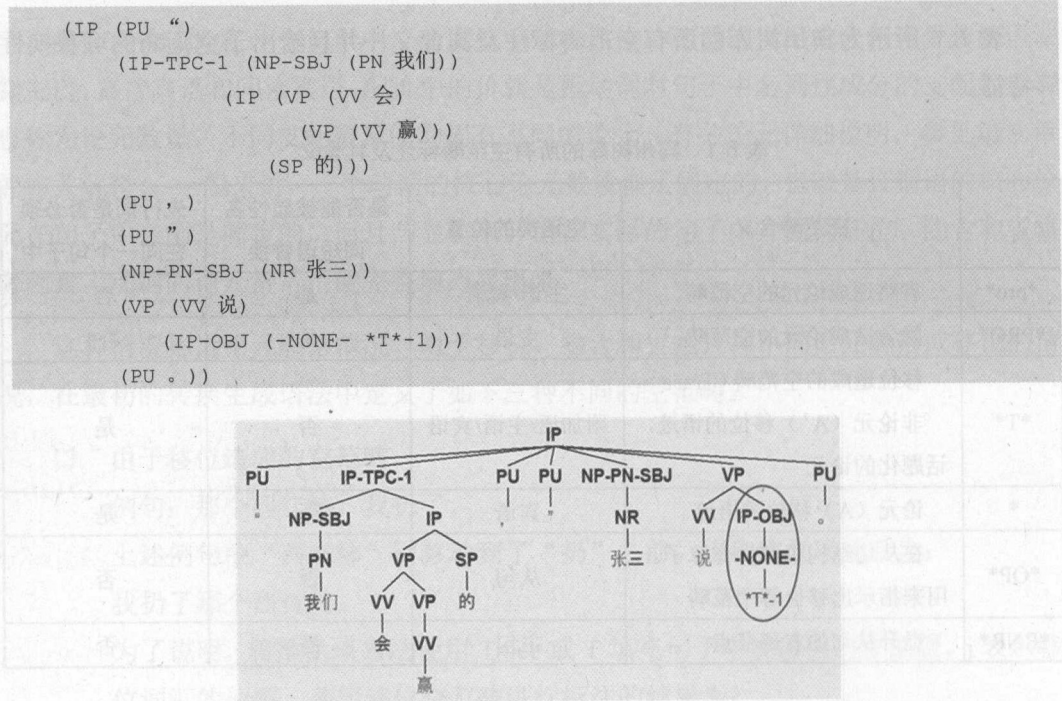


图 6.12 A'移位语迹——话题化

(2) (-NONE- *T*): 关系从句中的移位 (见图 6.13)。

```
(NP (CP (WHNP-1 (-NONE- *OP*))
  (CP (IP (NP-PN-SBJ (NR 日本))
    (VP (VV 发射)
      (NP-OBJ (-NONE- *T*-1))))
    (DEC 的)))
  (ADJP (JJ 大型))
  (NP (NN 科学)
    (NN 实验)
    (NN 卫星)))
```

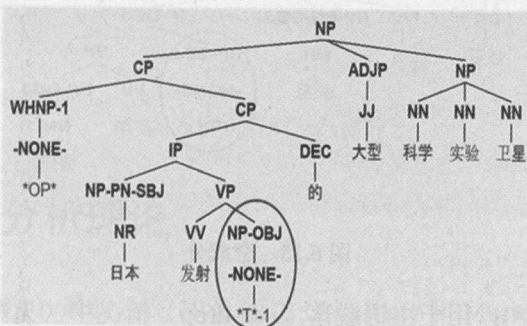


图 6.13 关系从句中的移位

(3) (-NONE- *): A-移位的语迹 (见图 6.14)。

```
(IP (NP-PN-SBJ-1 (NR 张三))
  (VP (VV 好像)
    (IP (NP-SBJ-2 (-NONE- *-1))
      (VP (SB 被)
        (VP (VV 打)
          (AS 了)
            (NP-OBJ (-NONE- *-2)))))))
```

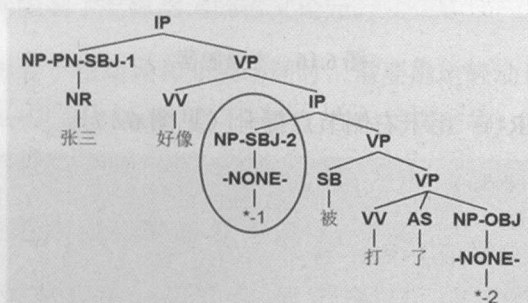


图 6.14 A-移位的语迹

(4) (-NONE- *PRO*): 在控制结构中的空成分 (见图 6.15)。

```
(IP (NP-PN-SBJ (NR 张三))
  (VP (VV 劝)
    (NP-PN-OBJ (NR 李四))
    (IP-OBJ (NP-SBJ (-NONE- *PRO*))
      (VP (VV 参加)
        (NP-OBJ (NN 会见))))))
```

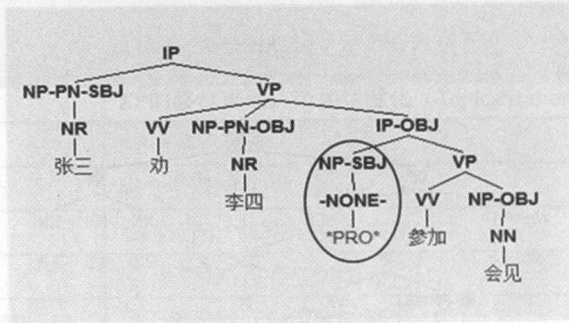


图 6.15 空成分

(5) (-NONE- *pro*): 用于主语脱落 (pro-drop) 情况中 (见图 6.16)。

```
(IP (NP-SBJ (-NONE- *pro*))
  (VP (VV 关上)
    (NP-OBJ (NN 门))))
(PU !)
```

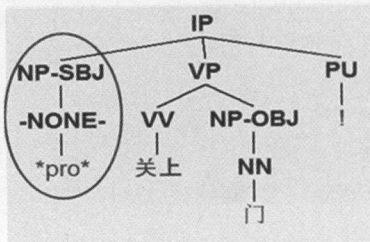


图 6.16 主语脱落

(6) (-NONE- *RNR*): 用于右侧节点提升 (见图 6.17)。

```
(VP (VP (ADVP (AD 积极))
  (VP (VV 引进)
    (NP-OBJ (-NONE- *RNR*-1))))
  (PU ,)
  (VP (ADVP (AD 精心))
    (V (VV 种养)))
```

(NP-OBJ-1 (ADJP (JJ 优稀))
 (ADJP (JJ 名贵))
 (NP (NN 品种))))))

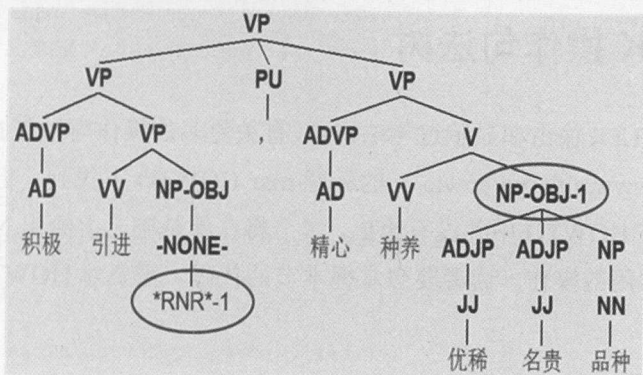


图 6.17 右侧节点提升

6.1.5 轻动词分析理论

轻动词 (Light Verbs) 最初是在 1988 年由 Grimshaw 和 Mester 提出来的, 他们认为在某些语言中存在一类特殊的动词——“轻动词”, 其在结构中所起的语义作用非常有限, 主要扮演功能上的辅助角色。20 世纪 90 年代中期, 轻动词假设被乔姆斯基吸收, 他把轻动词视为及物性谓语的核心。同时, 轻动词假设和空语类相结合, 轻动词的语义空灵, 甚至可以没有语音形式。乔姆斯基扩大了轻动词的外延。

本章所说的“轻动词”, 不完全与乔姆斯基的概念相同, 它更强调了在句法中的功能语义, 即“支撑”动词的概念。某些实义动词在某种特定的情况下, 可以作为轻动词使用。这些轻动词全部或部分地失去了原有词汇的语义内容, 作为述语句法功能的填充词, 支撑其后的名词化谓词。作为支撑动词, 轻动词必须与其后的名词化谓词共享至少一个核心论元。名词化谓词限制了支撑动词的语义论元。

当一个动词可同时用于轻动词和非轻动词时, 需要确定轻动词的语义。轻动词的论元结构和非轻动词 (Non-Light Verb) 是不同的。在轻动词结构, 真正的谓语通常是轻动词支持的名词化的谓词结构。

例 1: 他 经常 被邀请 到世界 著名 学府 作 讲演。

例 2: 企业 每逢 收购合并, 必然会 请 投资银行 作 顾问。

在例 1 中, “作”是一个支持名词化谓语“演讲”的轻动词, 而它本身没有什么意义,

并且没有自己的论元结构。“作”也可以用来作为非轻动词，如例 2，它的意义是“充当”，并且具有两个语义论元，“投资银行”和“顾问”。

6.1.6 NLTK 操作句法树

前文已经用 NLTK 输出和显示过句法树了，有关更多的操作可以在 NLTK 的 HOWTO 目录页面 (<http://www.nltk.org/howto/>) 的标题 tree HOWTO 下找到。但在实际操作中，有些常用的操作在 HOWTO 中并没有给出，本节将在此基础之上给出 NLTK 对句法树的一些更复杂、更常用的操作。读者要想掌握本节的代码，需要对 HOWTO 有一个初步的了解。

(1) 得到所有的叶子节点和词性。

函数：对于本节列出的所有函数读者都可以放到 `treelib.py` 文件下。

```
def flatten_childtrees(trees): # 扁平化子树结构
    children = []
    for t in trees:
        if t.height() < 3:
            children.extend(t.pos())
        elif t.height() == 3:
            children.append(Tree(t.label(), t.pos()))
        else:
            children.extend(flatten_childtrees([c for c in t]))
    return children

def flatten_deeptree(tree): # 解析 prop
    return Tree(tree.label(), flatten_childtrees([c for c in tree]))

def getwordposlist(tree):
    return [ tree[pos]+"/"+tree[pos[:-1]].label() for pos in tree.treepositions('leaves')]
```

执行代码如下。

```
# -*- coding: utf-8 -*-
import sys, os
from treelib import *

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
```

```

mytree = Tree.fromstring(u"(IP (IP (NP-SBJ (NN 建筑) (NN 公司)) (VP (VV 进)
(NP-OBJ (NN 区)))) (PU , ) (IP (NP-SBJ (ADJP (JJ 有关)) (NP (NN 部门))) (VP (ADVP (AD
先)) (VP (VV 送上) (NP-OBJ (DP (DT 这些)) (NP (NN 法规性) (NN 文件))))))
(PU , ) (IP (NP-SBJ (-NONE- *pro*)) (VP (ADVP (AD 然后)) (VP (VE 有) (IP-OBJ
(NP-SBJ (ADJP (JJ 专门)) (NP (NN 队伍))) (VP (VV 进行) (NP-OBJ (NN 监督) (NN 检
查)))))) (PU . )) ")

wordpostaglist=[word_pos[0][0]+"/"+word_pos[0][1] for word_pos in flatten_
deeptree(mytree).pos()]

for wordpostag in wordpostaglist:
    if wordpostag.find("-NONE-")==-1: #去除空范畴
        print wordpostag,
print
for wordpostag in getwordposlist(mytree):
    if wordpostag.find("-NONE-")==-1: #去除空范畴
        print wordpostag,

```

输出结果如下。

```

建筑/NN 公司/NN 进/VV 区/NN , /PU 有关/JJ 部门/NN 先/AD 送上/VV 这些/DT 法规性/NN 文件
/NN , /PU 然后/AD 有/VE 专门/JJ 队伍/NN 进行/VV 监督/NN 检查/NN 。 /PU
建筑/NN 公司/NN 进/VV 区/NN , /PU 有关/JJ 部门/NN 先/AD 送上/VV 这些/DT 法规性/NN 文件
/NN , /PU 然后/AD 有/VE 专门/JJ 队伍/NN 进行/VV 监督/NN 检查/NN 。 /PU

```

(2) 根据给定的词汇和子树标签, 得到词汇所在的一棵子树。

```

def getbranch(tree, keyword, branchlabel):
    gspos = tuple()
    for pos in tree.treepositions('leaves'):
        if tree[pos]==keyword: gspos = pos
    indx = -1
    for count in xrange(len(gspos)-1):
        if tree[gspos[:indx]].label() == branchlabel:
            return tree[gspos[:indx]]
    indx -= 1

```

执行代码如下。

```

# -*- coding: utf-8 -*-
import sys, os
import re
from treelib import *

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
mytree = Tree.fromstring(u"(IP (IP (NP-SBJ (NN 建筑) (NN 公司)) (VP (VV 进)

```

```
(NP-OBJ (NN 区)))) (PU , ) (IP (NP-SBJ (ADJP (JJ 有关)) (NP (NN 部门))) (VP (ADVP (AD
先)) (VP (VV 送上) (NP-OBJ (DP (DT 这些)) (NP (NN 法规性) (NN 文件))))))
(PU , ) (IP (NP-SBJ (-NONE- *pro*)) (VP (ADVP (AD 然后)) (VP (VE 有) (IP-OBJ
(NP-SBJ (ADJP (JJ 专门)) (NP (NN 队伍))) (VP (VV 进行) (NP-OBJ (NN 监督) (NN 检
查)))))) (PU 。)) ")
# 找到"公司"所在的 IP 子树
branch = getbranch(mytree, "公司", "IP")
print str(branch).decode("unicode-escape")
```

输出结果如下。

```
(IP (NP-SBJ (NN 建筑) (NN 公司)) (VP (VV 进) (NP-OBJ (NN 区))))
```

(3) 使用 ParentTree 过滤出所需要的子树 (消除递归嵌套)。

```
# -*- coding: utf-8 -*-
import sys, os
import re
from treelib import *

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
mytree = Tree.fromstring(u"(IP (IP (NP-SBJ (NN 建筑) (NN 公司)) (VP (VV 进)
(NP-OBJ (NN 区)))) (PU , ) (IP (NP-SBJ (ADJP (JJ 有关)) (NP (NN 部门))) (VP (ADVP (AD
先)) (VP (VV 送上) (NP-OBJ (DP (DT 这些)) (NP (NN 法规性) (NN 文件))))))
(PU , ) (IP (NP-SBJ (-NONE- *pro*)) (VP (ADVP (AD 然后)) (VP (VE 有) (IP-OBJ
(NP-SBJ (ADJP (JJ 专门)) (NP (NN 队伍))) (VP (VV 进行) (NP-OBJ (NN 监督) (NN 检
查)))))) (PU 。)) ")

ptree = ParentedTree.convert(mytree)
# 过滤出所有的 NP 短语 (最高层的 NP, 消除子节点也是 NP 的情况)
for subtree in ptree.subtrees(): # 递归遍历所有子树
    if subtree.label().find("NP") != -1 and mytree[subtree.treeposition()[:-1]].
label().find("NP") == -1:
        print str(subtree).decode("unicode-escape")
```

输出结果如下。

```
(NP-SBJ (NN 建筑) (NN 公司))
(NP-OBJ (NN 区))
(NP-SBJ (ADJP (JJ 有关)) (NP (NN 部门)))
(NP-OBJ (DP (DT 这些)) (NP (NN 法规性) (NN 文件)))
(NP-SBJ (-NONE- *pro*))
(NP-SBJ (ADJP (JJ 专门)) (NP (NN 队伍)))
(NP-OBJ (NN 监督) (NN 检查))
```


6.2 依存句法理论

依存句法是由法国语言学家泰斯尼耶尔(Lucien Tesnière)最早提出的。同时,他也是配价理论的奠基人。这两个概念都出现在他的鸿篇巨著——《结构句法基础》之中。因此,由于两者一脉相承的关系,我们将配价语法与依存句法放在同一章节(6.2节)来介绍。

依存句法代表了句法分析的另一个语法体系。与转换生成语法不同,依存语法和配价理论虽然诞生在法国,但发展壮大却是在德国。由于乔姆斯基的短语结构语法在解析德语时,精度明显低于英语,只能达到70%左右。这是由于德语中词序更为自由而导致的。为此迫使人们选择另一种语法体系,才开始注意到了依存句法理论。

一开始,人们对依存句法普遍有一种误解,认为它是短语结构文法的一种补充,但随着理论研究的深入,人们发现,依存句法不仅具有形式简洁、易于标注、便于应用等优点,更因为它与转换生成语法截然不同,完全是另一种句法体系。

Percival(1990)将依存关系分为两种:一种是句法依存,是指如果有两类元素,其中有一类元素只是在另一类出现时才会出现,那么就说前一类元素在句法上依存于后一类元素;另一种是语义依存,是指某些词的出现只是为了限定其他词的意义。

这与单纯从语法角度来分析句子的转换生成语法有了很大的不同。依存句法的依存关系中可以同时容纳句子的语法结构和语义结构的两种关系。近些年,认知科学的累累硕果使人们逐渐将目光从语法转向语义层面,希望两种标注并存的句法新理论对提高句子的识别精度会带来质的飞跃。

6.2.1 配价理论

配价理论源于一种形象的类比。泰斯尼耶尔把句子比喻成化学元素中的分子,分子是由原子按照一定的价结合到一起的。动词是句子的“中心”,所有的成分都是围绕着句子的中心动词展开的。那么,“中心”动词的价就决定了句子的成分结构。他称那些被中心动词直接支配的成分为“行动元”,那些起到辅助作用的成分为“状态元”。一个动词能够带的行动元个数就是它的价。为此他说:

“可以把动词比作一个带钩的原子,动词用这些钩子来吸引与其数量相同的行动元作为自己的从属成分。一个动词所具有的钩子的数量,即动词所能支配的行动元的数目,

就构成了我们所说的动词的配价。”

“应该指出的是，不必总是要求动词依照其配价带全所有的行动元，或者说让动词达到饱和状态。有些价可以不用或空缺。”

为此，他还作了形象的类比：动词代表一整出小戏剧，其中必然包括情节过程，大多数也包括人物和环境。如果将戏剧语言移用到配价理论中来，那么情节过程、人物和环境就分别成为动词、行动元和状态元。其中，动词表示情节过程。例如，句子“张三 在街上 碰见 李四”中，情节过程通过动词“碰见”表示出来；行动元指参与情节的人和事物，它通常由名词性词语充当，如上例中的“张三”、“李四”；状态元表示情节过程发生的时间、地点、方式等的环境，它由副词性词语充当；如上例中的“在街上”。动词是全句的支配成分，行动元则是动词的直接支配成分，而状态元是动词的辅助成分。

泰斯尼耶尔的“小戏”的说法很生动，它成为动词中心说和配价理论的经典比喻。动词的配价是配价语法的主体内容。早期的配价语法只包括动词的配价，现在，在计算语言学领域所指的配价也仍然以动词的配价为主。后来，泰斯尼耶尔在动词配价理论基础之上做了进一步发展，将配价理论发展到了其他类型的实词。更重要的是，在考察各种实词之间的支配关系时，泰斯尼耶尔逐渐形成了依存句法的思想。

确定一个动词的配价要求确定动词的价数、价质和价形。所谓价数就是指动词能够支配的配价成分的数量。根据动词的价数，将动词分为零价、一价、二价、三价等类型。所谓价质（价义）就是指动词的配价成分的语义性质（语义角色），如施事、受事、与事等。所谓价形就是指动词的各种配价成分能够出现的句法位置，如主语、宾语、状语、补语等位置。也就是动词与配价成分能够构成什么样的句式。

配价语法的概念并不复杂，它揭开了通向认知语义的大门。人们后来发现，认知语言学中的主体——背景理论，以及由此发展而来的意象图式理论与其有天然的联系。随着对配价理论的深入研究，人们提出了谓词论元的理论，把动词直接支配的行动元定义为核心论元，把动词间接支配的状态元定义为辅助论元。谓语动词及其核心论元和辅助论元最后都称为句子的语义角色，这区别于句子的语法成分，成为识别句义的一种新的标注系统。在本书的后面章节还会详细介绍。

需要特别说明的是，价数、价质和价形这种三位一体的识别模式，最近又引起了学界的重视，并以此为基本方法论，重新设计专门的算法来解决多年以来都难于处理的句子句法和语义识别的精度问题。

在中国，语言学界引入配价语法理论的时间始于 1978 年。30 多年来，汉语的语言

学和计算语言学的研究者对其进行了充分的研究,除发表了数百篇论文之外,还建立了较大规模的配价词典库。配价语法作为依存句法的理论基础早已成为汉语计算语言学的基础理论之一。

6.2.2 配价词典

配价理论一经产生之后,就面临着依据什么标准来确定动词价数这个问题。不同的学者提出了不同的标准。有的认为根据动词的意义来确定动词的配价,如动词“买”在语义上涉及4种必要成分:买者、卖者、买卖的物品和价格,那么“买”就是四价动词;有的认为根据成分的句法位置区分配价成分(必有论元)和非配价成分(可有论元)。早期的学者只将动词的主语和宾语看作配价成分,不能充当主语或宾语的成分不算配价成分。还有的根据成分的语义角色区分配价成分(必有论元)和非配价成分(可有论元)。早期的学者只承认施事(主体)、受事(客体)和与事三种语义成分为配价成分,其他语义成分都不算配价成分,等等,不一而足。这种矛盾来源于价数、价质和价形这三种指标的定义各不相同。

为了统一认识,人们开始从实际语料中寻求答案,这就导致了配价词典的产生。世界上第一部配价词典的作者是德国的语言学家 Helbig,他总结了构造配价词典条目的如下6个步骤:

(1) 分析动词对应的谓词的逻辑语义结构,找出形成完整谓词结构的可词汇化论元的数量。

(2) 标出动词具有的语义特征。

(3) 为动词标示语义格,也就是为第一步得到的那些论元赋予明确的语义角色,如施事、受事、地点、工具等。

(4) 对可词汇化的论元进行语义指称分析,并进行诸如 $[\pm\text{Anim}]$ 、 $[\pm\text{Hum}]$ 、 $[\pm\text{Abstr}]$ 之类的语义特征标识。

(5) 处理从语义层到句法层的映射问题,要考虑如下两种情况。

一是按照句子的功能成分,如主语、宾语等,二是按照句子成分的形态表示,如名词是什么格,介词短语的类型等,这是对行动元(补足语)的定性描述。

(6) 给定词项行动元(补足语)的定量描述,也就是给出动词项的价数,应区分必有和可有补足语。

Helbig 提出的确定配价的六原则模型要求在构造某种语言的配价词典（表）之前，要对动词进行细致的分类，要有一个语义格关系表，要有区别名词性成分的一套语义标记。此外，还要有一套适合该语言的句法关系集。

由此可见，按照 Helbig 的方法构建的语言词典是烦琐的，后人在此基础上进行了简化。就汉语而言，北大的配价词典最为著名。以北大的配价词典为例，看看汉语的配价词典的一些实例。

例：动词“吃”（见表 6.3）。

表 6.3 配价词典案例

Lexicon（词汇）	吃	Lexicon（词汇）	吃
pin_yin（拼音）	chi1	pin_yin（拼音）	chi1
sem_idx（义项序列）	1	sem_idx（义项序列）	2
semantics（语义）	—	semantics（语义）	受、挨
sem_taxonomy（语义类）	身体活动	sem_taxonomy（语义类）	其他行为
coord_valence（配价数）	2	coord_valence（配价数）	2
Subject（主语）	人	Subject（主语）	人
Object（宾语）	食物 药物	Object（宾语）	抽象事物
dative_role（格角色）	—	dative_role（格角色）	—
Example（例句）	吃食堂 / 吃药 / 吃靠山 / 吃力 / 吃在胃里 / 正吃着呢! / 吃坏了胃 / 这种菜吃了吃觉得很不错 / 野菜很吃油 / 吃软不吃硬	Example（例句）	吃亏 / 吃苦 / 吃惊 / 你先吃上一拳

注：访问网址：http://ccl.pku.edu.cn/ccl_sem_dict/。本例的词条来源于最初的版本，由于最新版的语义词典做了更新，可能词项有所变化。

北大语义配价词典的基本信息如下。

- ❑ Lexicon（词汇）和 pin_yin（拼音）。词汇的基本信息：名称、汉语拼音。
- ❑ sem_idx（义项序列）和 semantics（语义）。同一词汇有不同的义项，词汇的配价也要根据义项的不同分别定义。
- ❑ sem_taxonomy（语义类）。指定词汇所属的语义类。
- ❑ coord_valence（配价数）和 Subject（主语）、Object（宾语）。使词汇的配价数与句法成分建立关联。

- ❑ dative_role (格角色): 定义词汇的语义格。
- ❑ Example (例句): 给出词汇在语料中的实例。

6.2.3 依存理论概述

所谓依存句法，需要理解的关键概念在于依存。泰斯尼耶尔认为句子中各个成分之间都存在着支配与从属的关系。处于支配地位的词称为支配词 (Head)，也称为核心词；处于被支配地位的词称为从属词 (Dependency)，也称为修饰词。他认为，句子的结构表现为各个构成成分之间的层层递进的从属关系，它的顶端就成为一个支配所有成分的“中心结”(根节点)。“中心结”在绝大多数的情况下是动词，也就是说，动词是句子的中心。这种思想显然来自配价理论。

而句子各个成分之间的支配关系与被支配关系是单向的。因此，就很自然地构成了一棵以动词为中心的句法树。与前面的章节相同，类似的实例（实例中的词性标签，使用 Ltp 的标注规则）如下。

张三/nh 参加/v 了/u 这次/r 会议/n

上述句子是分词、词性标注之后得到的结果，把它写成列表的形式，如表 6.4 所示。

表 6.4 结果的列表形式

0	张三	nh
1	参加	v
2	了	u
3	这次	r
4	会议	n

第一列为词索引，第二列为词汇，第三列为词性标注。

按照依存句法的要求，定义动词为根节点，在新加入的依存关系列中，给出根节点的序号为 0，如表 6.5 所示。

表 6.5 给出根节点的序号为 0

0	张三	nh	
1	参加	v	0
2	了	u	
3	这次	r	
4	会议	n	

按照词汇的依存关系，再依次将剩下节点的依存关系补全，如表 6.6 所示。

表 6.6 依次将剩下节点的依存关系补全

0	张三	nh	1
1	参加	v	0
2	了	u	1
3	这次	r	4
4	会议	n	1

除“这次”这个词依赖于会议之外，其他词都依赖于根节点。

最后，标识出每个依存弧的名称。注意，这里使用的 Ltp 3.3 的依存关系集（详见下文）。这样就完成了一棵句法依存树的绘制。依存句法的生成如表 6.7 所示。

表 6.7 依存句法的生成

0	张三	nh	1	SBV
1	参加	v	0	HED
2	了	u	1	RAD
3	这次	r	4	ATT
4	会议	n	1	VOB

最后，同时使用 NLTK 库和 LTP-cloud 的可视化模块来显示这棵句法树的树状图，如图 6.18 所示。

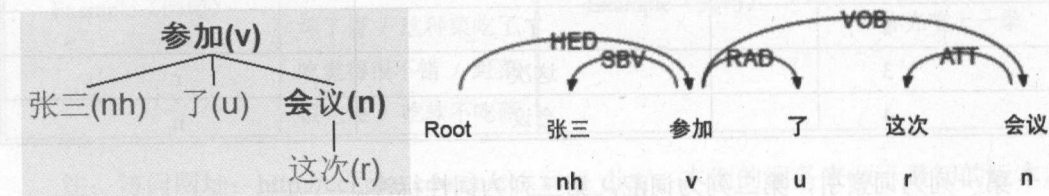


图 6.18 NLTK 和 Ltp 的依存句法树

图 6.18 中的两幅图中，NLTK 无法显示依存弧的方向和关系名称。而 LTP 的图形化显示不仅能够给出树状图，还给出了依存关系。这种句法树称为“依存树”（Dependency Tree）。

直观上，从上述流程中我们可以观察到依存语法与短语结构语法的相似之处，它们都将句子表示为一个树状的结构。所不同的是，依存树上连接词汇的每条弧代表了词汇之间的支配与从属的关系。“参加”这个根节点直接支配句子中除“这次”之外的其他词

汇。“这次”修饰了核心词“会议”，因此被“会议”支配。整个句子形成了一个三层的树状结构。

这是因为，依存关系是一种非对称的关系，这构成了层级的基础。因为一个元素可以支配和被支配，也因为每一个句子都是一个整体，所以句子可以被表示成这种树形结构。句子的中心称为根节点，句中所有其他元素都直接或间接地受根节点的支配。

同时还揭示出，句子是一个有组织的单位，其基本组成元素是词。联系产生了句子元素间的依存关系，所有这些联系就形成了句子的框架。因此，联系是概念联结的一种句法实现。同时，这种联系是说话人有意为之的（遣词造句）。也就是说，元素间的联系不仅要符合一定的规则，还要具有一定的语义。整个句子的意义不仅由句子的元素——词汇所确定，也要由各元素之间的联系所确定。联系使得用一个句子来表达一种完整的思想成为可能。

进一步来看，一个句子同时具有句法和语义结构。理解一个句子意味着掌握组成句子结构的所有联系。因为句法结构必须反映语义结构，所以句法结构和语义结构是平行的。而且，如同句法结构一样，语义结构也是二维的。

中国著名的计算语言学家冯志伟提出依存树应该满足如下5个条件。

- 单纯节点条件：在泰斯尼耶尔的“依存树”中，只有终极节点，没有非终极节点。也就是说，依存树中的所有节点所代表的都是句子中实际出现的具体的单词。当然，根据多标记的原理，依存树中的单词也可以是多标记的，再标以“词类”“静态语义特征”等信息。
- 单一父节点条件：在依存树中，除根节点没有父节点之外，所有的节点都只有一个父节点。父节点的方向指向子女节点，而不是相反。在数学上，这是一种非对称的、可传递的关系。
- 独根节点条件：一个依存树只能有一个根节点。这个根节点，也就是依存树中唯一没有父节点的节点，这个根节点支配着其他的所有的节点。
- 非交条件：依存树中的树枝不能彼此相交。
- 互斥条件：依存树中的节点之间，从上到下的支配关系和从左到右的前于关系是互相排斥的。也就是说，如果两个节点之间存在着支配关系，那么，它们之间就不能存在前于关系。

冯志伟提出的依存树这5个条件，更加形象地描述了依存树中各个节点之间的联系，显然更加直观、更加便于使用。

6.2.4 Ltp 依存分析介绍

CoNLL 是由 SIGNLL 组织的年度会议,有关会议的网址参见:<http://www.conll.org/>。该会议从 1997 年的第一届开始,到现在(2016 年)已经举办了 20 届,是 NLP 学界中最权威、最著名的研究会议。会议每年都发布一些最新的 NLP 研究成果,所有的论文都发布在 ACL Anthology (<http://www.aclweb.org/anthology/>) 上,可以自由下载。

与短语结构树一样,依存树也包含数据表示和图形表示两种结构。各种机构曾经都提供了依存句法树的数据表示,自然语言研究会(Conference on Natural Language Learning, CoNLL)也提供了自己的标注规范,CoNLL 会定期发布一些 NLP 的共享任务(Share Task),影响力比较大。CoNLL 规范已经成为一种表示语言分析结果的通用格式。

最初 CoNLL 标注格式一共包含 10 列,但实际使用的一般为 8 列,如表 6.8 所示。

表 6.8 CoNLL 标签说明

1	ID	当前词在句子中的序号,从 1 开始
2	FORM	当前词语或标点
3	LEMMA	当前词语(或标点)的原型或词干,在中文中,此列与 FORM 相同
4	CPOSTAG	当前词语的词性(粗粒度)
5	POSTAG	当前词语的词性(细粒度)
6	FEATS	句法特征,此列未被使用,全部以下划线代替
7	HEAD	当前词语的中心词
8	DEPREL	当前词语与中心词的依存关系

各家研究机构根据自己的需求都做了一些调整,以 Ltp-Cloud (Ltp 语言云) 上提供的依存句法系统为例,其数据标注规范如表 6.9 所示。

表 6.9 数据标注规范

列 号	含 义
1	单词在句子中的标号,从 0 开始
2	单词本身
3	空
4	空
5	单词词性标注信息
6	未登录词信息
7	依存句法关系中的父节点标号
8	依存句法关系类型

续表

列 号	含 义
9	空
10	空
11	如果单词是语义角色标注中的谓词，则为单词本身，否则为空
12	每个谓词占一行，每一列为该谓词的语义角色标注信息

如表 6.9 所示，在 Ltp 语言云的 CoNLL 格式中，分析结果的每一行代表句子中每个词的信息，词标号从 0 开始。分析结果的基础列有 10 列，之后的每一列代表文本中的语义信息，每列之间用 Tab 分割。此列值为空用"_"占位。下面给出 Ltp-Cloud 的 CoNLL 语言云的一个实例。

使用的例句与上例相同：张三参加了这次会议。

在浏览器地址栏输入：

[http://api.ltp-cloud.com/analysis/?api_key=YourApiKey&text=张三参加了这次会议。
&pattern=all&format=conll](http://api.ltp-cloud.com/analysis/?api_key=YourApiKey&text=张三参加了这次会议。&pattern=all&format=conll)

上述链接中的 YourApiKey 是用户注册 <http://www.ltp-cloud.com/> 网站后，系统生成的一个 api_key，每个用户的 api_key 都不相同。输入你自己的 YourApi_Key 之后，链接地址返回对例句的全部解析结果如图 6.19 所示。

0	张三	-	-	nh	S-Nh	1	SBV	-	-	-	(A0*)
1	参加	-	-	v	O	-1	HED	-	-	参加	(v*)
2	了	-	-	u	O	1	RAD	-	-	-	*
3	这次	-	-	r	O	4	ATT	-	-	-	(A1*
4	会议	-	-	n	O	1	VOB	-	-	-	*)
5	。	-	-	wp	O	1	WP	-	-	-	*

图 6.19 LTP 的 CoNLL 依存解析结果

最后，给出返回的 CoNLL 的数据在 pyltp 中的显示代码，输出结果中略去了空列、未登录词列和语义角色标注列，内容如下。

```
# -*- coding: utf-8 -*-
import sys,os
from pyltp import *

reload(sys)
sys.setdefaultencoding('utf-8')
```



```
words = "张三 参加 了 这次 会议 。".split(" ")
postagger = Postagger()
postagger.load("X:\\nltk_data\\ltp3.3\\pos.model")
postags = postagger.postag(words)

parser = Parser()
parser.load("X:\\nltk_data\\ltp3.3\\parser.model")
arcs = parser.parse(words, postags)
arclen = len(arcs)
conll = ""
for i in xrange(arclen):
    if arcs[i].head == 0:
        arcs[i].relation = "ROOT"
    conll += str(i)+"\\t"+words[i]+"\\t"+postags[i]+"\\t"+str(arcs[i].head-1)+"\\t"+arcs[i].relation+"\\n"
print conll
```

输出结果如下。

```
0   张三 nh   1   SBV
1   参加 v   -1  ROOT
2   了   u   1   RAD
3   这次 r   4   ATT
4   会议 n   1   VOB
5   。   wp  1   WP
```

LTP 的依存关系标签比较直观，主要识别句子中的“主、谓、宾、定、状、补”这些语法成分，并分析语言单位内成分之间的依存关系揭示其句法结构。LTP 的依存句法解析标签如表 6.10 所示。

表 6.10 LTP 的依存句法解析标签

关系类型	Tag	Description	Example
主谓关系	SBV	subject-verb	我送她一束花（我 <-- 送）
动宾关系	VOB	直接宾语, verb-object	我送她一束花（送 --> 花）
间宾关系	IOB	间接宾语, indirect-object	我送她一束花（送 --> 她）
前置宾语	FOB	前置宾语, fronting-object	他什么书都读（书 <-- 读）
兼语	DBL	double	他请我吃饭（请 --> 我）
定中关系	ATT	attribute	红苹果（红 <-- 苹果）
状中结构	ADV	adverbial	非常美丽（非常 <-- 美丽）
动补结构	CMP	complement	做完了作业（做 --> 完）

续表

关系类型	Tag	Description	Example
并列关系	COO	Coordinate	大山和大海（大山 --> 大海）
介宾关系	POB	preposition-object	在贸易区内（在 --> 内）
左附加关系	LAD	left adjunct	大山和大海（和 <-- 大海）
右附加关系	RAD	right adjunct	孩子们（孩子 --> 们）
独立结构	IS	independent structure	两个单句在结构上彼此独立
核心关系	HED	head	指整个句子的核心

6.2.5 Stanford 依存转换、解析

Stanford Parser 系统是一个可进行短语结构和依存结构分析的解析器，并提供了多种句法树解析算法，相比于 LTP Parser 对依存树的处理要复杂得多。Stanford 处理依存句法树的内容如下。

(1) 支持短语结构树转换为依存句法树。

```
package edu.stanford.nlp.ademo;

import java.util.Collection;

import edu.stanford.nlp.parser.lexparser.LexicalizedParser;
import edu.stanford.nlp.trees.GrammaticalStructure;
import edu.stanford.nlp.trees.GrammaticalStructureFactory;
import edu.stanford.nlp.trees.Tree;
import edu.stanford.nlp.trees.TreebankLanguagePack;
import edu.stanford.nlp.trees.TypedDependency;

public class PCFGDEPDemo {

    public static void main(String[] args) {
        LexicalizedParser lp = LexicalizedParser.loadModel("models/lexparser/
chinesePCFG.ser.gz"); //加载模型

        Tree parse = Tree.valueOf("(ROOT (IP (NP (NR 张三)) (VP (VV 参加) (AS 了)
(NP (DT 这次) (NN 会议))))))"); //字符串形式的短语结构树

        TreebankLanguagePack tlp = lp.getOp().langpack();
        GrammaticalStructureFactory gsf = tlp.grammaticalStructureFactory();
        GrammaticalStructure gs = gsf.newGrammaticalStructure(parse);

        // 依存关系，以 CoNLL 的形式显示
```

```

        System.out.println(GrammaticalStructure.dependenciesToString(gs,
            gs.typedDependencies(), parse, true, false));
        // 依存关系, 以 stanford 常规的形式显示
        Collection<TypedDependency> tdl = gs.allTypedDependencies();
        System.out.println(tdl);
        System.out.println();
    }
}

```

输出结果如下。

```

1   张三   _   NR   NR   _   2   nsubj   _   _
2   参加   _   VV   VV   _   0   root    _   _
3   了     _   AS   AS   _   2   asp    _   _
4   这次   _   DT   DT   _   5   dep    _   _
5   会议   _   NN   NN   _   2   dobj   _   _

```

```

[nsubj(参加-2, 张三-1), root(ROOT-0, 参加-2), asp(参加-2, 了-3), dep(会议-5, 这次-4),
dobj(参加-2, 会议-5)]

```

注意, 根节点序号从 0 开始, 而不是 LTP 的-1。

(2) 可以使用多种算法得到依存句法树, 这里提供基于 LSTM 神经网络依存解析器的测试代码。

```

package edu.stanford.nlp.ademo;

import java.io.StringReader;
import java.util.Collection;
import java.util.List;

import edu.stanford.nlp.ling.HasWord;
import edu.stanford.nlp.ling.TaggedWord;
import edu.stanford.nlp.parser.nndep.DependencyParser;
import edu.stanford.nlp.process.DocumentPreprocessor;
import edu.stanford.nlp.tagger.maxent.MaxentTagger;
import edu.stanford.nlp.trees.GrammaticalStructure;
import edu.stanford.nlp.trees.TypedDependency;

public class NNDepDemo {

    public static void main(String[] args) {
        String modelPath = "models/parser/nndep/CTB_ConLL_params.txt.gz";
    }
}

```



```
String taggerPath = "models/pos-tagger/chinese-distsim/chinese-distsim.
tagger";

String text = "张三 参加 了 这次 会议";

MaxentTagger tagger = new MaxentTagger(taggerPath);
DependencyParser parser = DependencyParser.loadFromModelFile(modelPath);

DocumentPreprocessor tokenizer = new DocumentPreprocessor(new StringReader
(text));

for (List<HasWord> sentence : tokenizer) {
    List<TaggedWord> tagged = tagger.tagSentence(sentence);
    GrammaticalStructure gs = parser.predict(tagged);
    Collection<TypedDependency> tdl = gs.allTypedDependencies();
    System.out.println(tdl);
}
}
```

输出结果如下。

```
[VMOD(参加-2, 张三-1), root(ROOT-0, 参加-2), VMOD(参加-2, 了-3), NMOD(会议-5, 这次
-4), OBJ(参加-2, 会议-5)]
```

(3) 给出复杂的 Stanford 依存句法标注体系。

该标注体系的完整内容列于 <http://universaldependencies.org/docs/u/dep/index.html> 中。用户可以参照网页具体的内容来了解每种标签的意义。一些常用的依存标签的中文说明如下。括号内为例子词汇。

中心语为谓词。

Subj——主语。

nsubj——名词性主语 (nominal subject) (同步, 建设)。

top——主题 (topic) (是, 建筑)。

npsubj——被动型主语 (nominal passive subject), 专指由“被”引导的被动句中的主语, 一般是谓词语义上的受事 (称作, 镍)。

csbj——从句主语 (clausal subject), 中文不存在。

xsubj——x 主语, 一般一个主语下面含多个从句 (完善, 有些)。

中心语为谓词或介词。

obj——宾语。

dobj——直接宾语（颁布，文件）。

iobj——间接宾语（indirect object），基本不存在。

range——间接宾语为数量词，又称为与格（成交，元）。

pobj——介词宾语（根据，要求）。

lobj——时间介词（来，近年）。

中心语为谓词。

comp——补语。

ccomp——从句补语，一般由两个动词构成，中心语引导后一个动词所在的从句（IP）（出现，纳入）。

xcomp——x 从句补语（xclausal complement），不存在。

acompl——形容词补语（adjectival complement）。

tcomp——时间补语（temporal complement）（遇到，以前）。

lccomp——位置补语（localizer complement）（占，以上）。

rscomp——结果补语（resultative complement）。

中心语为名词。

mod——修饰语（modifier）。

pass——被动修饰（passive）。

tmod——时间修饰（temporal modifier）。

rcmod——关系从句修饰（relative clause modifier）（问题，遇到）。

numod——数量修饰（numeric modifier）（规定，若干）。

ornmod——序数修饰（numeric modifier）。

clf——类别修饰（classifier modifier）（文件，件）。

nmod——复合名词修饰（noun compound modifier）（浦东，上海）。

amod——形容词修饰 (adjective modifier) (情况, 新)。

advmod——副词修饰 (adverbial modifier) (做到, 基本)。

vmod——动词修饰 (verb modifier, participle modifier)。

prnmod——插入词修饰 (parenthetical modifier)。

neg——不定修饰 (negative modifier) (遇到, 不)。

det——限定词修饰 (determiner modifier) (活动, 这些)。

possm——所属标记 (possessive marker), NP。

poss——所属修饰 (possessive modifier), NP。

dvpm——DVP 标记 (dvp marker), DVP (简单, 的)。

dvpmmod——DVP 修饰 (dvp modifier), DVP (采取, 简单)。

assm——关联标记 (associative marker), DNP (开发, 的)。

assmod——关联修饰 (associative modifier), NP|QP (教训, 特区)。

prep——介词修饰 (prepositional modifier) NP|VP|IP (采取, 对)。

clmod——从句修饰 (clause modifier) (因为, 开始)。

plmod——介词性地点修饰 (prepositional localizer modifier) (在, 上)。

asp——时态标词 (aspect marker) (做到, 了)。

partmod——分词修饰 (participial modifier) 不存在。

etc——等关系 (etc) (办法, 等)。

中心语为实词。

conj——联合 (conjunct)。

cop——系动 (copula) 双指助动词。

cc——连接 (coordination), 指中心词与连词 (开发, 与)。

其他。

attr——属性关系 (是, 工程)。

cordmod——并列联合动词 (coordinated verb compound) (颁布, 实行)。

mmod——情态动词 (modal verb) (得到, 能)。

ba——“把”字关系。

tc Claus——时间从句 (以后, 积累)。

cpm——补语化成分 (complementizer), 一般指“的”引导的 CP (振兴, 的)。

6.3 PCFG 短语结构句法分析

有关句法分析的算法, 全书给出两种不同的算法思想, 分别用于处理短语结构分析和依存句法。在短语结构语法中, 目前最成熟、精度最高的算法是 PCFG 算法。该算法的思想是基于概率图模型的, 前文讲过有关概率图模型的算法理论。因此, 本节主要介绍该算法的详细内容。

另一种基于依存句法理论的分析算法, 目前最新、最高效的是基于转换 (Transition-Based) 的 LSTM 算法, 最新版的 LTP 依存句法分析和 Stanford 依存句法分析都实现了该算法。该算法的思想是基于深度学习算法理论的, 在后面将详细介绍。

6.3.1 PCFG 短语结构

完全基于乔姆斯基的短语结构文法推导得到的结果, 往往得到很多不精确甚至有各种歧义的结果, 难以保证计算精度的要求。为此, 人们做了大量的研究, 有的选择其他的句法思想, 有的则通过开发树库资源来调整算法。前文介绍过的宾州树库就是基于人工标注的短语结构树库。树库资源提供了两个很重要的优势: 一方面可以从现有的大规模语料中学习出新规则; 另一方面, 可以很容易地获得每条规则概率的信息。而后者就发展成了 PCFG 短语结构算法基础。

PCFG 即 Probabilistic CFG, 也称为 Stochastic CFG, 直观的意义就是基于概率的短语结构分析。前文讲到, 乔姆斯基的短语结构文法表示为一个四元组: $G = (X, V, S, R)$ 。而在 PCFG 中增加了一个概率信息, 变为如下五元组。

$$PCFG = (X, V, S, R, P)$$

X 是一个有限词汇的集合 (词典)。它的元素称为词汇或终结符。

V 是一个有限标注的集合,叫作非终结符集合。它的元素称为变量或非终结符集合。

$S \in V$, 称为文法的开始符号。

R 是有序偶对 (α, β) 的集合, 也就是产生的规则集。

P 代表每个产生规则的统计概率。

如果把 “ \rightarrow ” 看作一个运算符, PCFG 可以写成如下形式。

形式: $A \rightarrow \alpha, P$

约束: $\sum_{\alpha} P(A \rightarrow \alpha) = 1$

约束的含义:
$$\left. \begin{array}{l} NP \rightarrow NN\ NN, \quad 0.80 \\ NP \rightarrow NN\ CC\ NN, 0.20 \end{array} \right\} \sum P = 1$$

下面根据一个例子来看 PCFG 求解最优句法树的过程。

有一个规则集, 内容如下。

- | | |
|-------------------------------------|---|
| $S \rightarrow NP\ VP, \quad 1.00$ | $NP \rightarrow \text{astronomers}, 0.10$ |
| $NP \rightarrow NP\ PP, \quad 0.40$ | $NP \rightarrow \text{saw}, \quad 0.04$ |
| $VP \rightarrow VP\ PP, \quad 0.30$ | $V \rightarrow \text{saw}, \quad 1.00$ |
| $PP \rightarrow P\ NP, \quad 1.00$ | $NP \rightarrow \text{telescopes}, 0.1$ |
| $VP \rightarrow V\ NP, \quad 0.70$ | $P \rightarrow \text{with}, 1.00$ |
| | $NP \rightarrow \text{ears}, 0.18$ |
| | $NP \rightarrow \text{stars}, 0.18$ |

给定句子 S : *Astronomers saw stars with ears.*

经过 CFG 推导, 得到两棵句法树, 如图 6.20 所示。

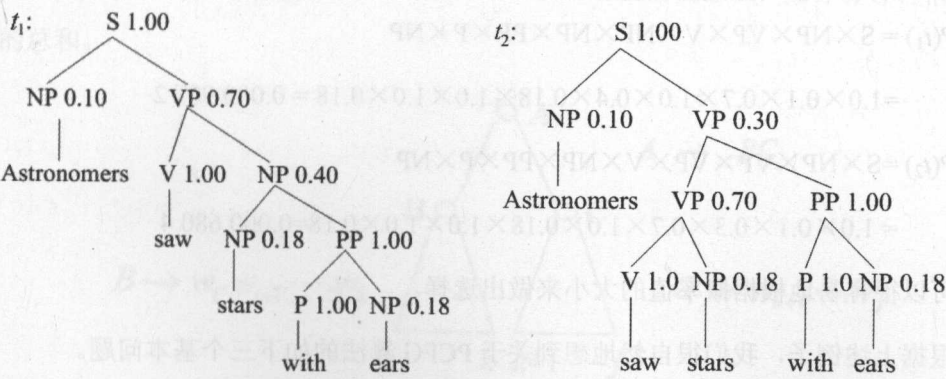


图 6.20 CFG 推导出的两棵子树

我们的目标是要找到哪一棵树才是最优的选择。

在计算之前，给出分析树概率的基本假设，内容如下。

- 位置不变性：子树的概率与其管辖的词在整个句子中所处的位置无关，即对于任意的 k , $P(A_{k(k+c)} \rightarrow w)$ 一样。
- 上下文无关性：子树的概率与子树管辖范围以外的词无关，即 $P(A_{kl} \rightarrow w | \text{任何超出 } k \sim l \text{ 范围的上下文}) = P(A_{kl} \rightarrow w)$ 。
- 祖先无关性：子树的概率与推导出该子树的祖先节点无关，即 $P(A_{kl} \rightarrow w | \text{任何除 } A \text{ 以外的祖先节点}) = P(A_{kl} \rightarrow w)$ 。

图 6.21 所示为三种基本假设下的计算过程。

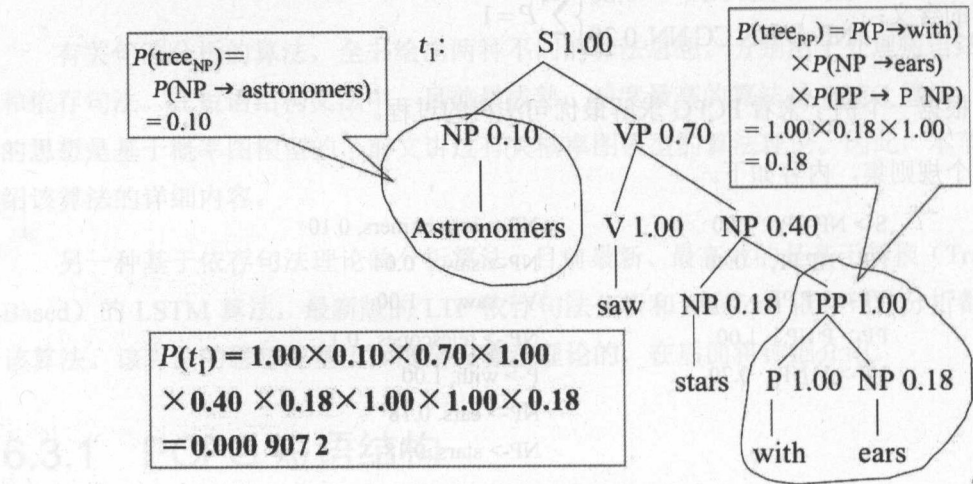


图 6.21 三种基本假设下的计算过程

依据这三个假设，分别得到两棵子树的概率，内容如下。

$$\begin{aligned} P(t_1) &= S \times NP \times VP \times V \times NP \times NP \times PP \times P \times NP \\ &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.000\ 907\ 2 \end{aligned}$$

$$\begin{aligned} P(t_2) &= S \times NP \times VP \times VP \times V \times NP \times PP \times P \times NP \\ &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.000\ 680\ 4 \end{aligned}$$

可以很容易地根据概率值的大小来做出选择。

根据上述例子，我们很自然地想到关于 PCFG 算法的如下三个基本问题。

- 给定句子 $W=w_1, w_2, \dots, w_n$ 和 PCFG G ，如何快速计算得到 $P(W|G)$ ？

- 给定句子 $W=w_1, w_2, \dots, w_n$ 和 PCFG G , 如何选择最佳的句法树?
- 给定句子 $W=w_1, w_2, \dots, w_n$ 和 PCFG G , 如何估计 G 的参数, 使得 $P(W|G)$ 最大?

6.3.2 内向算法和外向算法

1. 内向算法

使用内向算法解决第一个问题。内向算法的基本思想是从给定字符串的底层向上逐次推导出句子的全部概率。使用动态规划的方法计算由非终结符 A 推导出的某个字符片段 w_i, w_{i+1}, \dots, w_j 的概率 $\alpha_{ij}(A)$ 。语句 $W=w_1, w_2, \dots, w_n$ 的概率即为文法 $G(S)$ 中 S 推导出的字符串的概率 $\alpha_{1n}(S)$ 。

定义: 内向变量 $\alpha_{ij}(A)$ 是由非终结符 A 推导出的语句 W 中子字符串 w_i, w_{i+1}, \dots, w_j 的概率。

$$\alpha_{ij}(A) = P(A \Rightarrow w_i, w_{i+1}, \dots, w_j)$$

计算 $\alpha_{ij}(A)$ 的递推公式得到:

$$(1) \alpha_{ii}(A) = P(A \rightarrow w_i). \quad (\text{一步直接推导得出})$$

$$(2) \alpha_{ij}(A) = \sum_{B, C \in V_N} \sum_{i \leq k \leq j} P(A \rightarrow BC) \alpha_{ik}(B) \alpha_{(k+1)j}(C). \quad (\text{需要经过中间步骤得出})$$

如图 6.22 所示, 当 $i=j$ 时, 字符串 w_i, w_{i+1}, \dots, w_j 只有一个词 w_i , 由 A 推导出 w_i 的概率就是产生式 $A \rightarrow w_i$ 的概率 $P(A \rightarrow w_i)$; 当 $i \neq j$ 时, 也就是说, 字符串 w_i, w_{i+1}, \dots, w_j 中至少有两个词。根据约定, A 要推导出该字符串, 必须首先运用产生式 $A \rightarrow BC$ 。那么, 可用 B 推导出前半部 w_i, w_{i+1}, \dots, w_k , C 推导出后半部 $w_{k+1}, w_{k+2}, \dots, w_j$ 。由这一推导过程产生的概率为: $P(A \rightarrow BC) \alpha_{ik}(B) \alpha_{(k+1)j}(C)$ 。考虑到 B 、 C 和 k 取值的任意性, 应计算各种情况下概率的总和。

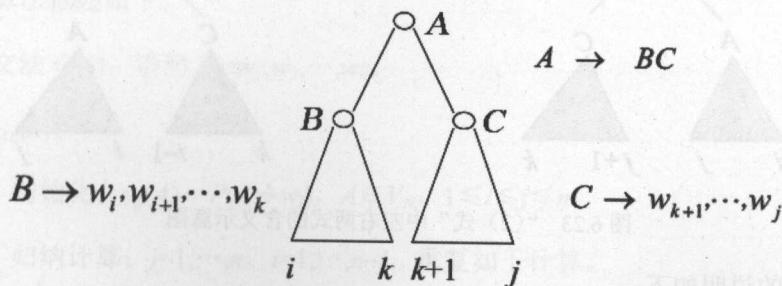


图 6.22 递推公式示意图

内向算法描述如下。

(1) 输入：文法 $G(S)$ ，语句 $W=w_1, w_2, \dots, w_n$ 。

(2) 输出： $P(S \Rightarrow^* w_1, w_2, \dots, w_n)$ 。

步骤 1，初始化： $\alpha_{ij}(A) = P(A \rightarrow w_i) \quad A \in V_N, 1 \leq i \leq j \leq n$

步骤 2，归纳计算： $\alpha_{i(i+j)}(A) = \sum_{B, C \in V_N} \sum_{i \leq k \leq i+j} P(A \rightarrow BC) \alpha_{ik}(B) \alpha_{(k+1)(i+j)}(C)$

步骤 3，终结： $P(S \Rightarrow^* w_1, w_2, \dots, w_n) = \alpha_{1n}(S)$

2. 外向算法

使用外向算法解决第一个问题。外向算法的基本思想是从给定字符串的顶层向下逐次推导出句子的概率。

定义：外向变量 $\beta_{ij}(A)$ 是由文法初始符号 S 推导出语句 $W=w_1, w_2, \dots, w_n$ 的过程中，到达扩展字符串 $w_1, \dots, w_{i-1}, A, w_{j+1}, w_n$ 的概率 ($A=w_i, \dots, w_j$)。

$$\beta_{ij}(A) = P(S \Rightarrow^* w_1, \dots, w_{i-1}, A, w_{j+1}, w_n)$$

计算 $\beta_{ij}(A)$ 的递推公式得到：

(1) $\beta_{1n}(A) = \delta(A, S)$ 。 (初始化)

(2) $\beta_{ij}(A) = \sum_{B, C} \sum_{k > j} P(B \rightarrow AC) \alpha_{j+1, k}(C) \beta_{ik}(B) + \sum_{B, C} \sum_{k < i} P(B \rightarrow CA) \alpha_{k, i-1}(C) \beta_{kj}(B)$ (推导

树是一棵二叉树)。

图 6.23 所示为“(2)式”中左右两式的含义示意图。

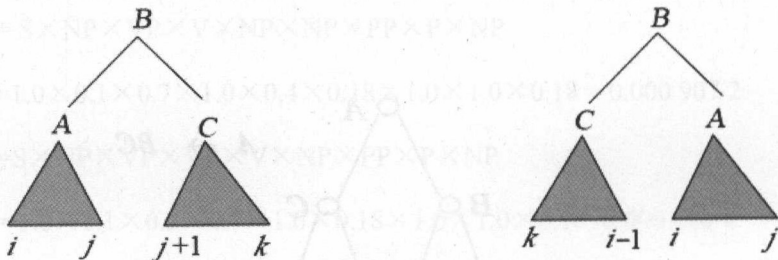


图 6.23 “(2)式”中左右两式的含义示意图

图 6.23 的说明如下。

(1) 当 $i=1, j=n$ 时，即 w_i, w_{i+1}, \dots, w_j 是整个语句时，由于语法中不可能有规则 $S \rightarrow A$,

因此, S 推导出 W 的过程中, 如果 $A \neq S$, 那么 A 推导出 W 的概率为 $0(\beta_{1n}(A))$ 。如果 $A=S$, 那么 $\beta_{1n}(A)$ 就是由初始符 S 推导出 W 的概率。因此, $\beta_{1n}(A)=1$ 。

(2) 当 $i \neq 1$ 或 $j \neq n$ 时, 如果在 S 推导出 W 的过程中, 出现了字符串 $w_1, \dots, w_{i-1}, A, w_{j+1}, \dots, w_n$, 则该推导过程必定使用了规则 $B \rightarrow AC$ 或 $B \rightarrow CA$ 。如果运用了规则 $B \rightarrow AC$, 则该推导可以分解为如下内容。

- 由 S 推导出 $w_1, \dots, w_{i-1}, B, w_{k+1}, \dots, w_n$, 其概率为 $\beta_{ik}(B)$ 。
- 运用产生式 $B \rightarrow AC$ 扩展非终结符 B , 其概率为 $P(B \rightarrow AC)$ 。
- 由非终结符 C 推导出 w_{j+1}, \dots, w_k , 其概率为 $\alpha_{j+1,k}(C)$ 。

外向算法描述如下。

(1) 输入: 文法 $G(S)$, 语句 $W=w_1, w_2, \dots, w_n$ 。

(2) 输出: $\beta_{ij}(A)$, $A \in N$, $1=i=j=n$ 。

步骤 1, 初始化: $\beta_{1n}(A) = \delta(A, S)$, $A \in N$ 。

步骤 2, 归纳计算: j 从 $n-1$ 到 0 , i 从 1 到 $n-j$, 重复计算如下。

$$\beta_{i(i+j)}(A) = \sum_{B, C} \sum_{i+j < k \leq n} P(B \rightarrow AC) \alpha_{i+j+1, k}(C) \beta_{ik}(B) + \sum_{B, C} \sum_{1 \leq k < i} P(B \rightarrow CA) \alpha_{k, i-1}(C) \beta_{k(i+j)}(B)$$

6.3.3 Viterbi 算法

第二个问题是如何选择最佳的句法树, 其解决方法使用前文讲过的 Viterbi 算法。在短语结构文法中, Viterbi 的定义如下。

Viterbi 变量 $\gamma_{ij}(A)$ 是由非终结符 A 推导出的语句 W 中字符串 w_i, w_{i+1}, \dots, w_j 的最大概率。变量 $\psi_{i,j}$ 用于记忆字符串 $W=w_1, w_2, \dots, w_n$ 的 Viterbi 语法分析结果。

Viterbi 算法描述如下。

输入: 文法 $G(S)$, 语句 $W=w_1, w_2, \dots, w_n$ 。

输出: $\gamma_{1n}(S)$ 。

步骤 1, 初始化: $\gamma_{ij}(A) = P(A \rightarrow w_i)$, $A \in V_N$, $1 \leq i \leq j \leq n$ 。

步骤 2, 归纳计算: $j=1, \dots, n$, $i=1, \dots, n-j$, 重复如下计算。

$$\gamma_{i(i+j)}(A) = \max_{B, C \in V_N; i \leq k \leq i+j} P(A \rightarrow BC) \gamma_{ik}(B) \gamma_{(k+1)(i+j)}(C)$$

$$\psi_{i(i+j)}(A) = \max_{B, C \in V_N, i \leq k \leq i+j} P(A \rightarrow BC) \gamma_{ik}(B) \gamma_{(k+1)(i+j)}(C)$$

步骤 3, 终结: $P(S \Rightarrow w_1, w_2, \dots, w_n) = \gamma_{1n}(S)$ 。

6.3.4 参数估计

如果有大量已标注语法结构的训练语料, 则可通过语料直接计算每个语法规则的使用次数。已知训练语料中的语法结构, 记录每个语法规则的使用次数, 用最大似然估计计算 PCFG 的参数, 即

$$P^*(N^j \rightarrow \zeta) = \frac{C(N^j \rightarrow \zeta)}{\sum_{\gamma} C(N^j \rightarrow \gamma)}$$

一般来讲, 树库的标注需要投入较大的人力, 时间和资金投入都比较大。而且, 树库也不可能覆盖所有的词汇信息, 在 NLP 中, 使用有限的资源预估目标的概率分布, 这种情况是常态。所谓参数估计, 就是对未知现象出现概率的一种计算方法。PCFG 中常用的方法是 EM (Expectation Maximization) 算法。

EM 算法描述: 初始时随机地给这些参数赋值, 得到语法 G_0 , 依据 G_0 和训练语料, 得到语法规则使用次数的期望值, 以期望次数运用于最大似然估计, 得到语法参数的新的估计, 由此得到新的语法 G_1 , 由 G_1 再次得到语法规则的使用次数的期望值, 然后又可以重新估计语法参数。循环这个过程, 语法参数将收敛于最大似然估计值。

PCFG 中使用的方法称为内向、外向算法: 给定 CFG G 和训练数据 $W=w_1, w_2, \dots, w_n$, 语法规则 $A \rightarrow BC$ 的使用次数的期望值如下。

$$\begin{aligned} C(A \rightarrow BC) &= \sum_{1 \leq i \leq k \leq j \leq n} P(A_{ij}, B_{ik}, C_{k+1,j} | w_1, \dots, w_n, G) \\ &= \frac{1}{P(w_1, \dots, w_n | G)} \sum_{1 \leq i \leq k \leq j \leq n} P(A_{ij}, B_{ik}, C_{k+1,j}, w_1, \dots, w_n | G) \\ &= \frac{1}{P(w_1, \dots, w_n | G)} \sum_{1 \leq i \leq k \leq j \leq n} \beta_{ij}(A) P(A \rightarrow BC) \alpha_{ik}(B) \alpha_{(k+1),j}(C) \end{aligned} \quad (1)$$

解释: 给定了语句, PCFG G 中产生式 $A \rightarrow BC$ 被用于产生的使用次数的期望值为: 在所有可能的 $1 \leq i \leq k \leq j \leq n$ 的情况下, w_1, w_2, \dots, w_n 的语法分析结构中, w_i, \dots, w_k 由 B 导出, w_{k+1}, \dots, w_j 由 C 导出, w_i, \dots, w_j 由 A 导出的概率总和。

类似的, 语法规则 $A \rightarrow a$ 的使用次数的期望值如下。

$$\begin{aligned}
& C(A \rightarrow a) \\
&= \sum_{1 \leq i \leq n} P(A_{ii} | w_1, \dots, w_n, G) \\
&= \frac{1}{P(w_1, \dots, w_n | G)} \sum_{1 \leq i \leq n} P(A_{ii}, w_1, \dots, w_n | G) \\
&= \frac{1}{P(w_1, \dots, w_n | G)} \sum_{1 \leq i \leq n} \beta_{ii}(A) P(A \rightarrow a) \delta(a, w_i)
\end{aligned} \tag{2}$$

G 的参数可由如下公式重新估计。

$$P^*(A \rightarrow \mu) = \frac{C(A \rightarrow \mu)}{\sum_{\mu} C(A \rightarrow \mu)} \tag{3}$$

其中, μ 要么为终结符号, 要么为两个非终结符号串, 即 $A \rightarrow \mu$ 为乔姆斯基语法范式要求的两种形式。

内向、外向算法的步骤如下。

步骤 1, 初始化: 随机地给 $P(A \rightarrow \mu)$ 赋值, 使得 $\sum_{\mu} P(A \rightarrow \mu) = 1$, 由此得到语法 G_0 。

令 $i=0$ 。

步骤 2, EM 步骤如下。

① 由 G_i 根据公式 (1) 和公式 (2), 计算期望值 $C(A \rightarrow BC)$ 和 $C(A \rightarrow a)$ 。

② M-步骤: 用 E-步骤所得的期望值, 根据公式 (3) 重新估计 $P(A \rightarrow \mu)$, 得到语法 G_{i+1} 。

步骤 3, 循环计算 $i=i+1$, 重复 EM 步骤, 直至 $P(A \rightarrow \mu)$ 收敛。

6.3.5 Stanford 的 PCFG 算法训练

在 Stanford PCFGParser 的 LexicalizedParser 类中, 有一个 Main 方法。它定义了训练外部语料的一些指令和参数, 用户可以从命令行访问指向。该 Main 方法可以用于构建和序列化树库的数据, 数据来源可以是文件或 URL, 一气呵成地训练和测试一个树库的解析结果 (主要用于解析器质量测试)。

从一个树库的目录中训练一个解析器, 并将训练结果存储为序列化语法文件, 需要执行如下指令。

```
java -mx1500m edu.stanford.nlp.parser.lexparser.LexicalizedParser [-v] -train
```

```
trainFilePath [fileRange] -saveToSerializedFile serializedGrammarFilename
```

主要参数如下。

- ❑ [-v]: 这里省略了一些参数细节。限于篇幅的关系, 仅给出必需的参数。
- ❑ -tLPP: 选定训练树库的语种类, 中文选择: `edu.stanford.nlp.parser.lexparser.ChineseTreebankParserParams`。
- ❑ `trainFilePath`: 输入的训练文件目录。
- ❑ `[fileRange]`: 训练文件范围为 0~1 000 个文件。
- ❑ `serializedGrammarFilename`: 输出序列化文件名。

从一个树库的目录中训练一个解析器(但不存储为序列化文件), 并提供测试结果报告, 需要执行如下指令。

```
java -mx1500m edu.stanford.nlp.parser.lexparser.LexicalizedParser [-v] -train
trainFilePath [fileRange] -testTreebank testFilePath [fileRange]
```

- ❑ `testFilePath`: 测试文件路径。
- ❑ `[fileRange]`: 测试文件范围。

在训练过程中, 如果序列化文件(`serializedGrammarFilename`)的路径以“.gz”结尾, 那么写入时就执行 GZip 压缩。如果序列化文件路径是一个 URL, 以 `http://` 开头, 则解析器就从这个 URL 中读取。`[fileRange]` 指定了一个数字值, 而这个值必须包含在文件名中(这非常适用于目前大多数的树库)。它的格式诸如“200-2199”或“1-300”, “500-725”, “9000”或者“1”(如果所有的树都在一个文件中, 可以省略此参数或只给一个伪参数, 如 0)。如果解析的文件名是“-”, 那么解析器从标准输入解析。

解析器除提供序列化的模型文件之外, 还可以提供一个模型的文本版本, 以提供用户审阅, 需要执行如下指令。

```
java edu.stanford.nlp.parser.lexparser.LexicalizedParser [-v] -train trainFilePath
[fileRange] [-saveToSerializedFile grammarPath] [-saveToTextFile grammarPath]
```

- ❑ `saveToTextFile`: 文本格式的模型文件路径。

关于 Stanford Parser 的更详细的内容可以在如下路径中找到。

<http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/parser/lexparser/LexicalizedParser.html#main-java.lang.String:A->

最后, 用 python 代码来实现 Stanford Parser 的训练过程, 回顾一下第 1 章, 实现了一个名为 `StanfordParser` 的 Python 类, 现在在这个类上做一些修改, 修改后的代码如下。


```

class StanfordParser(StanfordCoreNLP):
    def __init__(self, jarpath):
        StanfordCoreNLP.__init__(self, jarpath)
        self.modelpath = "" # 模型文件路径
        self.classfier = "edu.stanford.nlp.parser.lexparser.LexicalizedParser"
        self.opttype = ""

    def __buildcmd(self): # 构建命令行
        self.cmdline = 'java -mx500m -cp "'+self.jarpath+'" '+self.classfier+'
-outputFormat "'+self.opttype+'" '+self.modelpath+' '

    def parse(self, sent): # 解析句子
        self.savefile(self.tempsrcpath, sent)
        tagtxt = os.popen(self.cmdline+self.tempsrcpath, "r").read() # 输出到
                                                                    # 变量中
        self.delfile(self.tempsrcpath)
        return tagtxt

    def tagfile(self, sent, outpath): # 输出到文件
        self.savefile(self.tempsrcpath, sent)
        os.system(self.cmdline+self.tempsrcpath+' > '+outpath)
        self.delfile(self.tempsrcpath)

    def __buildtrain(self, trainpath, parsemodel): # 创建模型文件
        self.trainline = 'java -mx2g -cp "'+self.jarpath+'" '+self.classfier+'
-tLPP edu.stanford.nlp.parser.lexparser.ChineseTreebankParserParams -train "'+trainpath
+'"' -saveToSerializedFile "'+parsemodel+'"'

    def __buildtraintxt(self, trainpath, parsemodel, txtmodel): # 创建模型文件
        self.trainline = 'java -mx2g -cp "'+self.jarpath+'" '+self.classfier+'
-tLPP edu.stanford.nlp.parser.lexparser.ChineseTreebankParserParams -train "'+
trainpath+'"' -saveToSerializedFile "'+parsemodel+'"' -saveToTextFile "'+txtmodel+'"'
    def trainmodel(self, trainpath, parsemodel, txtmodel=""): # 训练模型
        if txtmodel:
            self.__buildtraintxt(trainpath, parsemodel, txtmodel)
        else:
            self.__buildtrain(trainpath, parsemodel)
        os.system(self.trainline)
        print "save model to ", parsemodel

```

在__init__(self, jarpath)构造方法的参数中, 仅保留了 jarpath 这一个参数, 去掉其他两个参数。在类的内部也做了相应的修改。之后, 增加了两个训练方法: __buildtraintxt 和 __buildtrain。__buildtraintxt 用以输出文本格式和压缩序列化格式的模型文件,

`_buildtrain` 仅用于输出序列化格式的压缩文件。

它们都由 `trainmodel` 这个方法调用。具体调用根据输入参数的不同自动判断。

`Trainmodel` 有如下三个参数。

- ❑ `trainpath`: 训练文件路径。
- ❑ `parsemodel`: 训练后的模型。
- ❑ `txtmodel`: 训练后文本形式的模型。

外部调用代码如下。

```
# -*- coding: utf-8 -*-
import sys,os
from stanford import *

# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')
os.environ['JAVA_HOME'] = 'D:\\Java7\\jdk1.8.0_65\\bin\\java.exe' # 配置环境变量

root = "E:/nltk_data/stanford-corenlp/" # 库所在路径
trainpath = "trainfile/" # 宾州树库的样例文件 chtb_0001.mrg (utf-8)
modelpath = "trainmodel.ser.gz"
txtmodelpath = "trainmodel.ser"
parser=StanfordParser(root)
result = parser.trainmodel(trainpath ,modelpath,txtmodelpath)
# result = parser.trainmodel(trainpath ,modelpath)
print result
```

可以看到，训练后在根目录下生成两个模型：`trainmodel.ser.gz` 和 `trainmodel.ser`。训练阶段输出的部分日志如下。

```
LexicalizedParser invoked on Sun Jul 31 07:44:36 CST 2016 with arguments:
  -tLPP edu.stanford.nlp.parser.lexparser.ChineseTreebankParserParams -train
trainfile/ -saveToSerializedFile trainmodel.ser.gz -saveToTextFile trainmodel.ser
Training a parser from treebank dir: trainfile/
Reading trees...done [read 11 trees]. Time elapsed: 0 ms
Options parameters:
...
Using ChineseTreebankParserParams chineseSplitDouHao=false chineseSplitPunct=true
chineseSplitPunctLR=false markVVsisterIP=true markVPadjunct=true chineseSplitVP=3
mergeNNVV=false unaryIP=false unaryCP=false paRootDtr=false markPsisterIP=true
```

```

markIPsisterVVorP=true markADgrandchildOfIP=false gpaAD=true markIPsisterBA=true
markNPmodNP=true markNPconj=true markMultiNtag=false markIPsisDEC=true markIPconj=
false markIPadjsbj=false markPostverbalP=false markPostverbalPP=false baseNP=false
headFinder=levy discardFrgs=false dominatesV=false
  Binarizing trees...done. Time elapsed: 13 ms
  Extracting PCFG...done. Time elapsed: 180 ms
  Extracting Lexicon...ChineseUWM: treating unknown word as the average of their
equivalents by first-character identity. useUnicodeType: false
  ChineseUWM: using Good-Turing smoothing for unknown words.
  Total tokens: 416.0
  Total WordTag types: 236
  Total tag types: 35
  Total word types: 227
done. Time elapsed: 151 ms
  Compiling grammar...done Time elapsed: 1 ms
  Extracting Dependencies...done. Time elapsed: 78 ms
Done training parser.
Writing parser in text grammar format to file trainmodel.ser.....done.
Writing parser in serialized format to file trainmodel.ser.gz done.
Grammar States  TagsWords  UnaryR  BinaryR  Taggings
Grammar 194 35 228 45 252 236
ParserPack is edu.stanford.nlp.parser.lexparser.ChineseTreebankParserParams
Lexicon is edu.stanford.nlp.parser.lexparser.ChineseLexicon
Options parameters:
...
save model to trainmodel.ser.gz

```

最后, 给出文本的模型文件片段, 内容如下。

```

...
BEGIN LEXICON edu.stanford.nlp.parser.lexparser.ChineseUnknownWordModel
"NR" -> "上海" SEEN 4.0
"NR" -> ".*" SEEN 19.0
"NR" -> "浦东" SEEN 10.0
"NN" -> ".*" SEEN 128.0
"CC" -> "与" SEEN 1.0
"VV" -> "开发" SEEN 1.0
"CC" -> ".*" SEEN 8.0
"NN" -> "开发" SEEN 4.0
"VV" -> ".*" SEEN 48.0
".$$." -> ".$$" SEEN 11.0
"NN" -> "建设" SEEN 5.0
".$$." -> ".*" SEEN 11.0

```


"NN" -> "法制" SEEN 3.0

6.4 结语

本章从理论、算法和实现三个方面介绍了传统 NLP 最核心的任务：句法解析。在理论部分，本节介绍了目前最为流行的两大句法分析理论：转换生成语法和依存句法。

转换生成句法是在 20 世纪 50~60 年代创立的一种句法理论。该理论是建立在乔姆斯基的语言学理论之上的重要的句法理论，转换生成句法的成熟并不是一蹴而就的，它经历了半个多世纪的发展才逐渐成熟起来。句法理论包含如下 4 个重要的方面：短语结构文法的规则与推导、转换生成句法中的汉语句类、空范畴理论、轻动词分析理论等。针对这 4 个部分，本章都做了详细的分析。在理论部分最后，使用 NLTK 程序库，以宾州树库为样本，提供几个操作句法树库的示例程序。通过上述示例程序，读者完全可以自由地操作树库，并根据自身的需求达成分析的目标。

本章介绍了基于概率的短语结构树分析算法：PCFG 算法。内容分为如下两大部分，第一部分是算法的基本原理，包括 PCFG 短语结构，内向算法和外向算法，Viterbi 算法及参数估计。第二部分讲解了通过 Stanford 的 PCFG 算法框架，如何训练自己的句法模型，并用于实践之中。

本章介绍的第二种句法理论是目前在汉语中应用更为广泛的依存句法理论。该理论脱胎于配价理论，为此，在介绍之前首先对配价理论做了简要的分析，并简要介绍了配价词典的结构。之后，介绍了配价理论的基本思想，并给出了一个使用依存句法原则手工分析句子的例子。

在实现部分介绍了 CoNLL 依存树文件格式，然后以两个依存句法解析器为例，来实现依存句法标注示例。

- ❑ LTP 依存句法解析器：标注规范、标注句子的实例。
- ❑ Stanford 依存句法标注器：StanfordNLP 依存标注规范、短语结构到依存句法的转换、Stanford 依存解析器的使用。

第 7 章

建设语言资源库

前面各章花了很多篇幅讨论了 NLP 的各种算法。本章将重点转到 NLP 算法所必须使用的语料库上。一方面，没有语料库，算法不能单独解析语言；另一方面，语料库的质量、规模等特征都对解析的结果产生很重要的影响。

与传统对语料库的分类不同，这里介绍了两类语料库：语法语料库和语义知识库。它们都可以作为算法的训练资源。除此之外，还介绍一种新兴的大规模语料库：百科知识库。目前对它的研究还处于初级阶段。虽然在某些方面获得了很大的突破，但是在自动语义解析方面仍面临诸多的难题。

7.1 语料库概述

由于电脑存储的容量大，文本资料真实，信息提取快速、准确。语言学家借助电子语料可以从多方面、多层次描写语言并验证各种语言理论和假设，甚至建立新的语言模式和语言观。因此，语料库也就应运而生了。

语料库 (Corpus, 复数为 Corpora 或 Corpuses) 被定义为：为语言研究和应用而收集的，在计算机中存储的语言材料，由自然出现的书面语或口语的样本汇集而成，用来代表特定的语言或语言变体。

——来自 2011 年出版的《语言学名词》。

语料库的出现是计算语言学和语言学发展的结果，也是信息社会的需要。

语料库并非语篇的简单堆砌或集合，它应具有如下几个基本特征。

- ☐ 样本代表性。
- ☐ 规模有限性。
- ☐ 机读形式化。

在 7.1.3 节将通过“国家语委语料库”的建设实例给出详细的说明。

7.1.1 语料库的简史

世界上最早的语料库是 1961 年，弗朗西斯（N. Francis）和库塞拉（H. Kucera）为首的一批语言学家和计算机专家汇集在美国的布朗大学，合作建成了世界上最早的机读语料库，即布朗语料库（Brown Corpus）。布朗语料库包含各种不同的文体，根据抽样调查决定了一个他们认为英文平衡语料库应有的分布，再根据此分布收集了共计 100 多个词的语料，并加上词性标注，由人工输入计算机。所以，布朗语料库确定了语料库的两个最早的特征：一个是使用计算机存储，并可以机读；另一个是为使语料具备代表性，语料的选择服从某种分布。这是语料库建设的最初特征，它们一直伴随着语料库的建设，并发展完善起来。

之后，同类的语料库逐渐发展起来。20 世纪 80 年代，以朗文语料库（Longman Corpus）为代表的语料库逐步走向商用，这使得语料库的建设走出学术界而进入商业市场，使中型语料库焕发了新的生命力。比起最早的布朗语料库，朗文语料库的规模显著扩大了，达到 5 000 万词级。朗文语料库由如下三个大的语料库组成，分别为：朗文/兰开斯特英语语料库（LLELC）、朗文口语语料库（LSC）、朗文英语学习者语料库（LCLE），其建设的主要目标之一是编纂英语学习词典，为外国人学习英语提供服务。

1992 年，美国宾夕法尼亚大学（University of Pennsylvania）语言资源联盟（Linguistic Data Consortium, LDC）宣布成立（<http://www ldc.upenn.edu/>）。这是第二代语料库的标准性事件。它的目的是构建、收集和发布用于研发的语音和文本数据库、词典及其他资源。该联盟提供了一种可供大规模发展和普遍的共享用于语言工程技术研究的资源的新机制，目前已经拥有超过 100 家公司、大学和政府机构会员单位。为 197 个会员机构和 458 个非会员机构提供数据。在众多的语言学资源中，LDC 的一个著名成果就是主持开发了宾州树库（UPenn Tree Bank, PTB）。该项目由宾夕法尼亚大学计算机系的 M.Marcus 主持，到 1993 年完成了近 300 万个词的英语句子的句法结构标注。

2000年,由LDC(语言数据协会)发行了UPenn的中文树库(CTB 1.0),最初规模较小,仅包含10万个词,4185句,到目前的8.0版,已经发展为7万多句,包含100万个词的大型汉语树库,成为研究汉语句法识别最重要、最权威的资源。

语料库在中国发展较晚,但由于互联网的发展,大规模、超大规模的网页资源成为具有代表性的汉语生语料库的数据来源。其中,最为著名的是搜狗互联网语料库(URL: <http://www.sogou.com/labs/resource/t.php>)。该语料库来源于互联网各种类型的1.3亿个原始网页,压缩前的大小超过了5TB。目前的版本是2008版,所有语料都可免费下载,用于研究如下内容。

- ☐ 迷你版(样例数据,61KB): tar.gz 格式、zip 格式。
- ☐ 完整版(1TB): (硬盘拷贝)。
- ☐ 历史版本(130GB): V 2.0 (硬盘拷贝)。

语料格式说明如下。

```
<doc>
<docno>页面 ID</docno>
<url>页面 URL</url>
页面原始内容(网页原始内容,保留了HTML标记)
</doc>
```

熟语料库方面,较为有名的汉语标注语料库都是在20世纪90年代之后发展起来的。在目前众多的语法语料库中,规模较大、代表性较强、质量较高、语料开放的有如下两个。

(1) 国家现代汉语语料库(URL: <http://www.cncorpus.org/index.aspx>)。该语料库是国家语委主持建立的一个现代汉语书面语通用平衡样本语料库,它于1993年开始建设。该语料库的第一批语料数据是1919—1992年的语料,共7000万字,以后每年递增1000万字,是目前最大的现代汉语平衡语料库。该语料库以语言文字的信息处理、语言文字规范和标准的制定、语言文字的学术研究、语文教育和语言文字的社会应用为主要服务对象。

(2) PFR语料库。由北京大学计算语言学研究所与富士通公司(Fujitsu)合作,以2700万字的1998年《人民日报》为源语料,手工加工、标注建立的语料库,加工项目包括词语切分、词性标注、专有名词(专有名词短语)标注(示例),并制定出《现代汉语语料库加工手册——词语切分与词性标注》。

对于上述两个熟语料库,下文都有介绍,这里不进行详细说明。

7.1.2 语言资源库的分类

传统上,语料库语言学根据选择的语料内容、选择的方式及建设目的的不同,将语料库的类型划分为如下 5 个部分。下面给出简要的说明。

- 通用性和专用性。通用语料库:又称为一般语料库,是文本的集合,为了保证收集的语料具有广泛的代表性,对语料采用系统的办法进行采集,用于事先未指定的语言学研究。通用语料库应有“平衡性”,即语料库要收集不同类型、不同领域的包括口头的或书面的文本。通用语料库也称为系统语料库或平衡语料库,有时还称为“核心语料库”。当然,严格说来,这些不同的名称之间还是存在差异的。专用语料库:又称为专门用途语料库(Special Purpose Corpus),指用于某种特殊研究的语料库。它又可分为方言语料库、区域性语料库、非标准语料库和初学者语料库等。它还可分为书面语料库和口语语料库。口语语料库是研究口语特征的重要工具,如语音语调的规律,其研究成果在语音合成中有重要应用。口语语料库的建设涉及口语真实语料的采集及语音转录,工作量极大。
- 异质性与同质性。异质语料库:大量收集文字材料,尽可能广泛地接受各类材料而没有事先制定任何选材原则。收藏的文本在格式和内容上各异,而存储的格式和原来的出版物完全一样。同质语料库:是“异质语料库”的对立面,一般用于专业语料库。
- 动态性和静态性。动态语料库:又称为监控语料库(Monitor Corpora),用于观察现代语言的变迁。与此相对的是静态语料库,只收集某一固定时期的共时语言材料,语料库建成后,就不再扩充。
- 共时性与历时性。共时语料库:指收集同一时代的语言使用样本构成的语料库。与此相对的是历时语料库,指收集不同时代的语言使用样本构成的语料库。历时语料库主要用来观察和研究语言的历时变化,共时语料库则用来观察和研究某一时代的语言使用状况。对历时语料库的分解可以得到多个共时语料库。
- 平行/双语语料库。把两种语言中完全对应的文本输入计算机,通过分析对比找出两者的对应关系,可用于机器翻译研究。近年来还出现了多语语料库,如可以从网上免费下载的 Europarl Parallel Corpus(European Parliament Proceedings Parallel Corpus)就收集了多达 11 种欧洲议会的多语言文集。

——来自《汉语语料库应用教程》

在实践中,同一语料库可以具备上述多种特性,这5个相互对立的特征揭示了一个重要的原则,即任何语料库的语料选择都是一种平衡性的结果。语料的选择要综合考虑上述5个维度的平衡。例如,小规模的通用语料库(通用分词语料库)添加了大量专业领域语料之后,可以变成专业语料库。静态语料库通过定期添加外部语料,变为动态语料库,等等。笔者认为上述各个要点作为特性来分析和评估语料库更为合适。而语料库的分类应该根据构建的目的不同而划分。目前,常见的语料库类别大概有两种:语法语料库和语义知识库。

(1) 语法语料库。语法语料库常作为NLP的基础资源,用于学习和训练NLP模型,如训练分词、命名实体、词性标注、句法解析、语义组块、论元角色等基础NLP任务的语料库。这些语料库的多数都来自影响面较大的大众媒体、书籍文献等语料,具有广泛性和代表性;语料的选择都经过精挑细选,构成上要求具有典型性,能够涵盖绝大多数的语言现象。例如,分词语料需要包含足够多的高频、常用词汇;句法树库必须涵盖绝大多数的汉语句型,等等。语法语料库开发出的熟语料会作为基础语言资源,最终训练成语言模型,因此对标注精度要求高。在开发过程中不可避免地需要大量人工标注或人工校对工作,除非是国家级项目,语法语料库容量都不大,源语料的容量一般不会超过1GB。

(2) 语义知识库。语义知识库包括如下两大类:早期手工建立的语义知识库,如HowNet、WordNet语义知识库。这类知识库依赖于手工标注,因此容量都不大。HowNet包含1500多个义原、10多万个常用词汇。董振东先生历时十年才开发完成。这类知识库在自然语言处理中可用于提供词汇的论元角色、上下位关系、语义消歧和相似度计算等。由于词汇量不大,其在实践中的应用范围也很有限。另一种是近些年逐渐流行起来的百科知识库。由于Google的Word2Vec算法的成功,人们通过算法可以直接训练出词汇间的语义相似度。Word2Vec的训练过程不依赖于手工标注,语义相似度正确率很高,开创了大规模非监督语义计算的先河。因此,百科知识库的建设目标是为NLP语义计算提供大规模、全覆盖的语义知识资源,有条件地兼顾推理运算的支持。与语法语料库相比,一般百科知识库的容量都比较大,最小的中文维基百科也有30GB。

后面章节会对这两种知识库进行详细的解读。

7.1.3 语料库的设计实例:国家语委语料库

无论是语法语料库还是语义知识库,它们的建设都是一个既耗费时间,又耗费金钱的过程。建设一个具有代表性的大型语料库需要多个团队、历时多年才能完成。因此,

在投资建设语料库之前，必须对语料库进行总体规划。规划一个语料库的方方面面可以从如下各个要素来综合考虑（见表 7.1）。

表 7.1 语料库设计要素

设计要素	属 性	属性特征
A.语料选取	规模	百万词集 千万词集 亿万词集 …
	领域	政治 经济 体育 科技 …
	体裁	文学 应用文 新闻 …
	时代	共时 历时 …
	语体	书面语 口语 …
	语种	单语 双语 多语 双语平行语料库 双语比较语料库
	语言层次	语音（音节、韵律） 语法（词、句）
B.加工方式	数据形式	TEXT文本 HTML文本 数据库 …
	编码形式	TEI标准 自定义编码标准
	加工层次	词性 句法 语义 语篇 …
	加工方式	自动 人机互助 人工
C.应用目标	应用领域	通用 词典编纂 机器翻译…
	应用工具	检索工具 人机界面 数据接口…

下面以国家语委现代汉语语料库为例，介绍语料库的主要内容和功能。

1. 国家语委现代汉语语料库概况

国家语委语料库于 1991 年 12 月国家语言文字工作委员会提出立项。被列为国家语委“九五”、“十五”科研重大项目，得到国家科技部“863”、“973”计划多个项目的支持。该语料库建设主要用于：语言文字的信息处理、语言文字规范和标准的制定、语言文字的学术研究、语文教育、语言文字的社会应用等。从 1998 年年底开始，最早建成 7 000 万字的生语料库；到 2015 年年底发布已完成 1 亿字的生语料和 5 000 万字的标注语料；该语料库建设和加工工作还在继续进行。

介绍该语料库的目的在于，可以将其作为用户自主开发语料库的一个范本，用于实际的 NLP 应用项目。下面介绍该语料库的基本概况。

(1) 选材的内容分布（见图 7.1）。

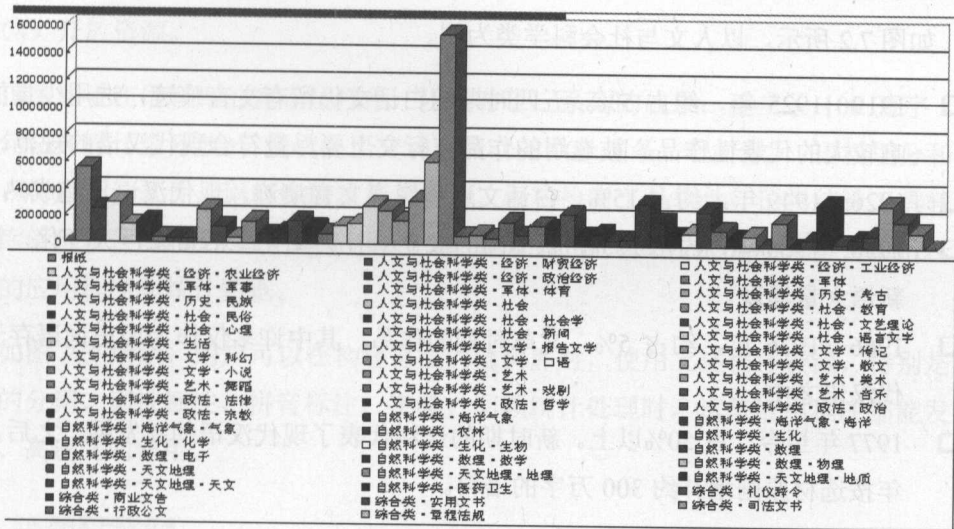


图 7.1 语料库选材的内容分布

图 7.1 显示了语料国家语委语料库的语料获取选材的内容分布信息。

- 人文与社会科学类。人文与社会科学类划分为 8 个大类和 30 个小类，约占语料总量的 50%。大类包括：政法、历史、社会、经济、艺术、文学、军体和生活。
- 自然科学类。自然科学划分为 6 类，占语料总量的 30%，包括：数理、生化、天文地理、海洋气象、农林和医药卫生。
- 综合类。综合类语料由应用文和难于归类的其他语料两部分组成。应用文主要分为 6 类，占语料总量的 20%，包括：行政公文、章程法规、司法文书、商业文告、礼仪辞令和实用文书。

(2) 选材的历时性分布（见图 7.2）。

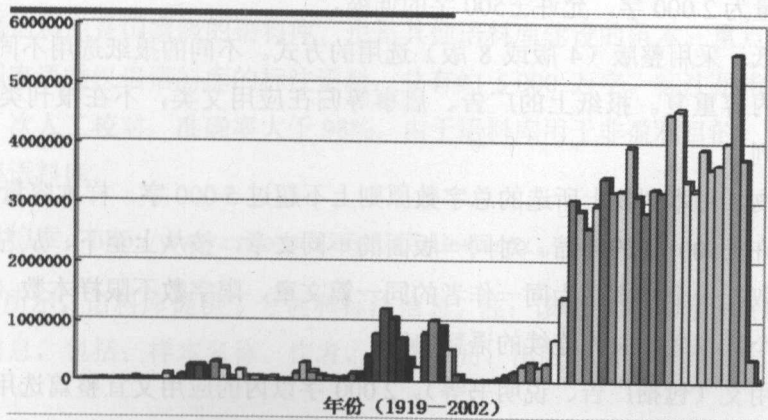


图 7.2 语料库选材的历时性分布

如图 7.2 所示,以人文与社会科学类为例。

- ❑ 1919—1925 年,约占 5%。五四时期的白话文仍留有文言痕迹,选用少量的影响较大的代表性作品。被选用的作品在行文上要尽量符合现代汉语的规范。
- ❑ 1926—1949 年,约占 15%。白话文逐步脱离文言痕迹,现代汉语日趋成熟。
- ❑ 1950—1965 年,约占 25%。新中国的成立给社会文化生活带来巨大变化,新词新语大量涌现。
- ❑ 1966—1976 年,约占 5%。文革时期的作品,其中许多仅作为历史词语存于现代汉语之中。
- ❑ 1977 年至今,占 50%以上。新时期的语料代表了现代汉语的新发展。之后,每年按选材原则增补约 300 万字的语料。

(3) 选材的抽样。

抽样原则如下。

- ❑ 语言材料的多样性。选用政论性文章、新闻报道、各类文学艺术作品、科普读物、通俗读物、学术专论及各种应用文语体等现代汉语作品。
- ❑ 语言材料的完整性。2 000 字以下的文章原则上全篇采用。报纸可采取整篇文章、整版和整张相结合的方式。
- ❑ 语言材料的遍历性。选材要注意各学科、各学科分支、各行各业,以及社会生活各个领域的语言文字应用的代表性。

抽样的数量与方式如下。

- ❑ 书籍。抽样数量一般占全书字数的 3%~5%,字数最多不超过 10 000 字。样本容量为 2 000 字,允许±500 字的伸缩。
- ❑ 报纸。采用整版(4 版或 8 版)选用的方式。不同的报纸选用不同的月份,以免内容重复。报纸上的广告、启事等归在应用文类,不在报刊类语料的统计之列。
- ❑ 刊物。每本刊物上所选的总字数原则上不超过 5 000 字。样本容量为 2 000 字,允许±500 字的伸缩。对同一版面的不同文章,按从上至下、从左到右的顺序选取。一个样本必为同一作者的同一篇文章,限字数不限样本数(报刊除外)。每个样本之中必为连续的语料内容。
- ❑ 应用文(包括广告、说明书等)。2 000 字以内的应用文宜整篇选用。对于篇幅较长的应用文,所选样本的容量为 2 000 字,允许±500 字的伸缩。

(4) 开放资源。

国家语委在语料库的建设过程中,还发布了一系列的成果:建立了100万字(5万句)句法树库(未开放)、现代汉语分词词表(88 000 词条,已开放)、语料切分和标注软件、树库标注软件、语料库校对加工软件、语料检索工具软件、语料统计工具软件、语料库管理软件等。感兴趣的读者可以下载(地址:<http://www.cncorpus.org/resources.aspx>)有关的应用程序和统计数据。

如图 7.3 所示,用户可以在构建自己的语料库时,使用上述程序资源,特别是在小规模的分词、词性标注、拼音标注、字频、词频统计处理时,这些应用程序都能发挥高精度、高速度的作用。

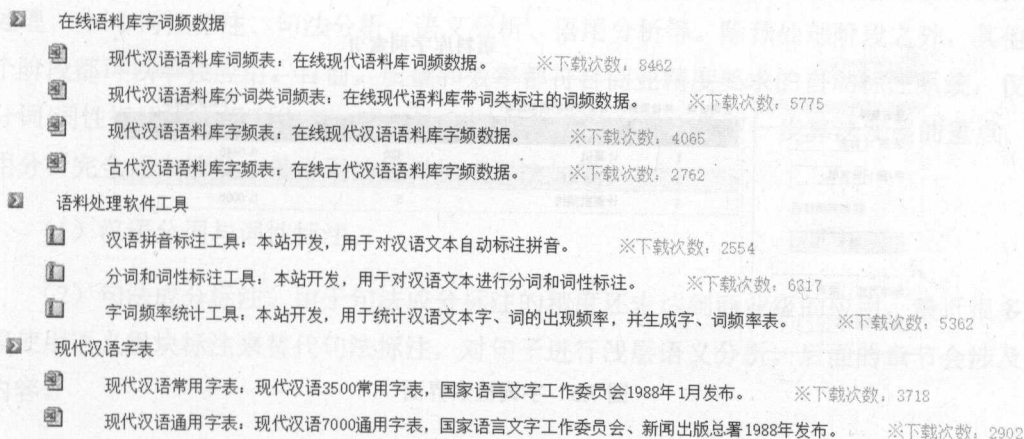


图 7.3 语委汉语语料库的应用程序和统计数据

2. 语委汉语语料库的功能

语委汉语语料库是国家级的语料库,也是其他语料库建设的范本。重点考察标注语料库部分。国家语委汉语语料库的标注语料,共有约5 000万字。标注是指分词和词类标注,历经3次人工校对,准确率大于98%。由于语料库用于非盈利目的,读者可以通过网络操作该语料库。

(1) 语料检索: <http://www.cncorpus.org/CCindex.aspx>。

如图 7.4 所示,语料库提供了分词和标注信息。注:该语料库的每一条语料都提供了语料来源信息,包括:样本名称、作者、写作时间、出版时间、书刊名称、编著者和出版社。

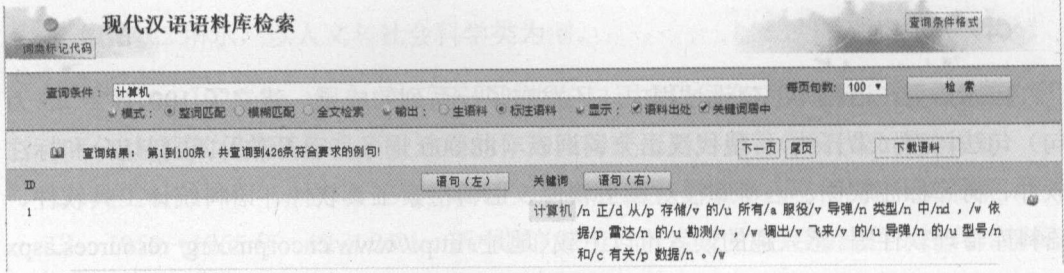


图 7.4 语料检索界面

(2) 字词检索: <http://www.cncorpus.org/wdindex.aspx>。

如图 7.5 所示, 语料库还提供了字词检索, 通过输入的关键词, 查询语料库中对应的词汇, 同时提供次数和词频信息。

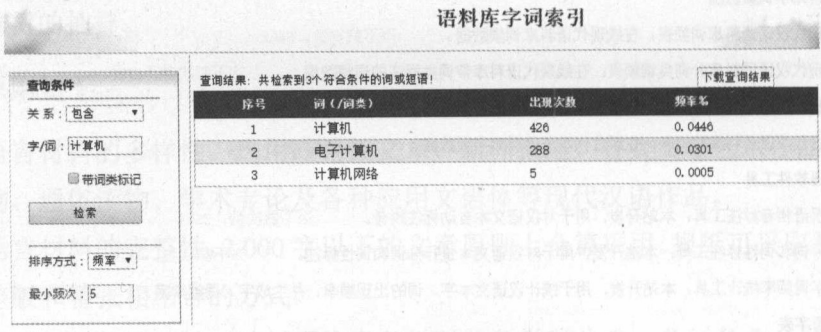


图 7.5 字词检索界面

(3) 分词/词性标注: <http://www.cncorpus.org/CpsParser.aspx>。

如图 7.6 所示, 该功能可根据输入的句子, 自动进行分词和词性标注。

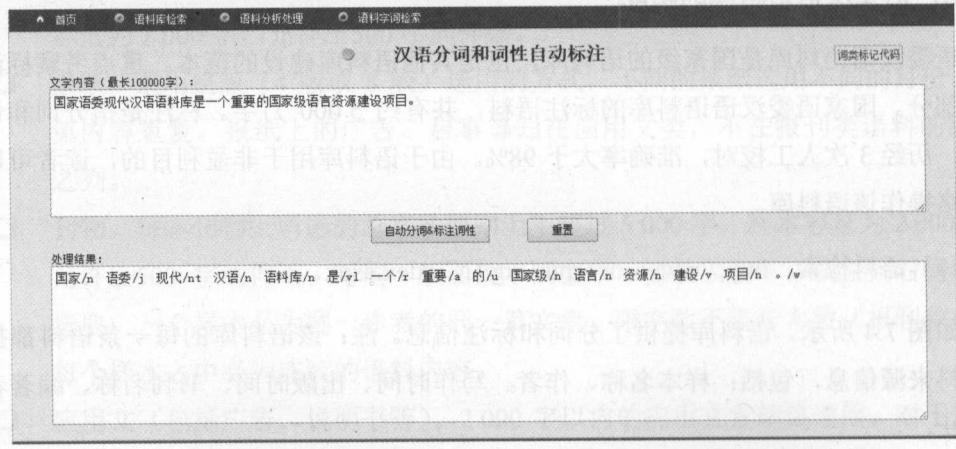


图 7.6 分词/词性标注界面

其他功能还包括,汉语拼音自动标注(<http://www.cncorpus.org/CpsPinyinTagger.aspx>)、字词频率统计(<http://www.cncorpus.org/CpsTongji.aspx>)等。这里不一一列举。

语料库的 Web 功能与前面介绍的语料切分和标注软件、语料库校对加工软件、语料检索工具软件、语料统计工具软件等功能都是一致的。读者可以通过下载应用程序,在本地对自己的语料进行处理。

7.1.4 语料库的层次加工

由于 NLP 对语言的解析是一个流程化的结构,流程中的每个节点都对应着相应的语料资源,这就构成了语料库的不同加工层次。语料库的加工可以包含如下几个层次:预处理、分词/词性标注、句法分析、语义分析、语用分析等。除预处理阶段之外,其他几个阶段都可以单独应用。目前,质量和效率都符合商业精度要求的自动标注系统,仅在分词/词性标注层次得到实现,而句法和语义分析两个层次是下一步算法发展的重点,语用分析完全还未涉及。最常用的语料加工层次如下。

(1) 汉语分词与词性标注。

(2) 句法成分标注。由于句法成分标注的精度还未达到商业级的应用,最近很多专家使用语义组块标注来替代句法标注,对句子进行浅层语义分析。后面的章节会涉及此内容。

(3) 语义角色标注。一般在句法成分标注之后进行,给每个句法成分标注上语义信息,如施事、受事等。

(4) 语用信息标注。就是对文本标注上相关的语用信息,如话题、述题、话轮、省略成分等,为语用分析服务。它可以在生语料的基础上进行,也可以在熟语料的基础上进行。

另外,在 4 个主层次之间,还可以根据应用系统需求的不同,添加辅助层次的标注,如命名实体识别、专有名词的标注、最大名词短语的标注、主题标引等。总之,研究者和使用者可以根据需求对标注好的主层次的任意一层进行进一步的标注,只要这种标注有助于达到最终的应用目标。

生语料库只有经过标注之后才能变成熟语料使用。标注方法有两种:人工标注和自动标注。如果自动标注器的精度不高,则只能使用人工标注的方法。如果使用人工加工、标注语料库,则需要遵循一些基本的原则。对此, G.Leech 曾提出了有标记的语料库应

满足如下基本原则。

- ❑ 所作标注可以删除，恢复到原始语料。
- ❑ 所作标注可以单独抽出，另处存储。
- ❑ 语料库的最终使用者应该知道标注原则和标注符号的意义。
- ❑ 标注规范，即标注所用标准的描述和解释性文档。
- ❑ 记录标注者、标注地点和怎样标注的文档。
- ❑ 由于标注通常会出现差错、不一致或歧义现象，因此应当有关于标注质量的说明。
- ❑ 在语料的使用说明中，应该说明标注是什么人用什么方法做的。
- ❑ 应向用户声明，语料标注并非绝对无误，它只是一种可能有用的工具。
- ❑ 标注模式应不依赖于某一家之言，尽可能中立。

事实证明，一方面，人工标注语料是一项枯燥、繁重，而又难于统一规范的任务；另一方面，人工标注的投入非常大、耗时很长、成本高昂，又不可避免地受到人为主观因素的影响，要想达到理想效果是一个极其艰难的过程。以宾州树库为例，整个项目到现在历时 10 多年，人工标注的句法树也不到 8 万句。虽然，这 7 万多句涵盖了大多数的汉语句法现象，但包含的汉语词汇也只有 5 万多个。

而在实践中，不同项目都有领域专业性。一般来讲，很少有人从零开始构建自己的语法语料库。更现实的做法是使用自动分词/标注器，自动标注专业语料。自动标注也分为两种情况：一种情况是已经具有领域词典；另一种情况是没有领域词典。如果开发者已经有领域词典，建议参考第 5 章的命名实体识别一节的相关算法策略，一般能达到 95% 左右的分词结果。

如果还没有领域词典，则需要一个训练专业语料库，形成专业语言模型的过程。以分词、词性标注为例，常用的自动标注方法是首先将专业语料按文件的容量分为若干个领域语料包。用一个高精度、通用性的分词/词性标注器对其中一个领域语料包自动标注。要求该分词/词性标注器的基础训练语料必须是开源的。然后对自动标注后的专业语料进行人工校对，找出错误，进行修正。校对工作最少由两人共同完成，一人负责找出错误，另一人负责审核。一份语料最少要校对两次，才能确保准确率不低于 98%。然后，将校对好的语料作为新增语料纳入到基础训练语料中。使用分词/词性标注器对新的语料库进行训练，这样就获得了第一个版本的领域专属分词模型。

接下来，使用新的分词模型自动标注下一个语料包，之后再校对、再添加、再训练，如此迭代，直至分词/词性标注器对所有语料包的分词结果精度高于预定的一个精度百分比，就算标注工作训练完成。

7.2 语法语料库

在构建 NLP 系统时,常要用到一套高精度的 NLP 算法库,以及一套高质量的、广为接受的基础语料资源。有关 NLP 的算法情况前面已经介绍过。下面重点谈谈基础语料的选择问题。

7.2.1 中文分词语料库

作为基础语料使用的中文分词库(含词性标注)在网上能找到的有很多。例如,LANCAST 中文语料库、浙大汉语译文语料库、搜狗 mini 语料库等。最常用、最著名的中文分词语料库共有两个:一个是 1998 年人民日报标注语料库(版本 1.0,下面简称 PFR 语料库);另一个是微软亚洲研究院中文分词语料库(下面简称 MSR 语料库)。

1. PFR 语料库

PFR 语料库是在得到人民日报社新闻信息中心许可的条件下,以 1998 年人民日报语料为对象,由北京大学计算语言研究所和富士通研究开发有限公司共同制作的标注语料库。该语料库对 600 多万字节的中文文章进行了分词及词性标注,其被作为原始数据应用于大量的研究和论文中。PFR 词性标注规范遵循《北大规范》,有关词性标签的详细内容可以参见第 3 章的相关内容或《人民日报标注语料库(PFR)使用说明书》。

PFR 语料库是纯文本文件,文件中每一行代表一个自然段或者一个标题,一篇文章有若干个自然段,因此在语料中一篇文章是由多行组成的。在一篇文章内的段落之间是不空行的。两篇文章之间会有一个空行,表示文章的分界线。同时,下一篇文章的"篇章号-段号"都会有所改变。

下面举例给出语料的样本。

19980101-01-001-001/m 迈向/v 充满/v 希望/n 的/u 新/a 世纪/n ——/w
一九九八年/t 新年/t 讲话/n (/w 附/v 图片/n 1/m 张/q)/w

语料的开头是编号。比如"19980101-01-001-001"表示这一自然段是 1998 年 1 月 1 日的第 01 版的第 001 篇文章的第 001 自然段,用短横线隔开的 4 部分按照顺序是"年月日-版本号-篇章号-段号"。标号也提供了词性标注,词性固定为"m(数词)"。

标号之后,是 2 个单字节空格,然后开始正文。正文部分按照规范已经切分成词,并且加上标注,标注的格式为"词语/词性",即词语后面加单斜线,再紧跟词性标记。词

与词之间用 2 个单字节空格隔开。每段最后的词，在标记之后也有 2 个单字节空格，保持格式一致。

我们对该语料做了一些精简，去掉了开头的编号，删除了段落间的空行。经过对全部 PFR 语料（1998 年 1 月~6 月，上半年）的统计，语料库中共有词汇 172 120 个（相同汉字但不同词性的词汇按词性划分为多个）。除去各种符号和数字、字母构成的词汇之外，完全由汉字构成的词汇共计 155 747 个。词频词典可从 <http://www.threedweb.cn/thread-1592-1-1.html> 下载分析。以 CRF 算法作为分词算法，PFR 语料库作为基础语料进行分词达到的精度在 94.7% 左右。

人民日报语料 1998 年 1 月的语料资源可从 <http://www.threedweb.cn/thread-1591-1-1.html> 免费下载。另有 1998 年上半年的收费版本，可从 <http://www.hyxr.com.cn/cp/qikan/19.htm> 购买。值得一提的是，有人在 Google 上发布了一个基于 CRF++ 的中文分词器，项目名为 Nlpbamboo，已经将 PFR 的 1998 年 1 月的语料训练成 CRF++ 的模型库。该库可以从 <http://www.threedweb.cn/thread-1319-1-1.html> 查询下载。

2. MSR 语料库

MSR 语料库是微软研究院开发的一套中文分词基础语料库。该语料库仅对词汇做了切分，而没有给出词性标注。MSR 语料库包含两套不同用途的文本：一套用于标准的词汇切分；另一套用于命名实体识别。读者可从 <http://www.threedweb.cn/thread-1593-1-1.html> 下载。下面给出该语料库的样例：

“人们常说生活是部教科书，而血与火的战争更是不可多得的教科书，她确实是名副其实的‘我的大学’。”

文档只有正文部分，每句都已经按照规范切分成词，词与词之间用空格分开，标点符号也用空格分开。与 PFR 语料库相同，句与句之间用换行符分隔。经过对 MSR 语料（分词部分）的统计，语料库中共有词汇 68 946 个。除去各种符号和数字、字母构成的词汇之外，完全由汉字构成的词汇共计 68 444 个。MSR 语料库及其词频词典也可从 <http://www.threedweb.cn/thread-1593-1-1.html> 下载分析。该贴子还包含一个基于 CRF 模板标签的版本 MSRA 语料库，主要是为了满足不同分词算法的要求。同样以 CRF 算法作为分词算法，MSRA 语料库作为基础语料进行分词达到的精度在 97% 以上。

3. 词频统计的实现

最后，给出 PFR 语料库的词频统计实现的 Python 代码，内容如下。


```

# -*- coding: utf-8 -*-

import sys, os
from framework import *
import nltk
import re
from cStringIO import StringIO

reload(sys)
sys.setdefaultencoding('utf8')

def fullfreq(fcontent, sumdict): # 全部词汇
    sent = " ".join(fcontent.splitlines()).strip().decode("utf8")
    sTuple=[ nltk.tag.str2tuple(t) for t in sent.split(" ") ]
    fredist=nltk.FreqDist(sTuple) #获取统计结果
    print len(fredist)
    for localkey in fredist:
        if localkey in sumdict : #检查当前词频是否在词典中存在
            sumdict[localkey]=sumdict[localkey]+fredist[localkey]
#如果存在, 将词频累加, 并更新词典值
        elif str(localkey[1]).find("None")==-1 :
            sumdict[localkey]=fredist[localkey] #将当前词频添加到词典中

hanzi= re.compile(ur"[\u4e00-\u9fa5]+") # 切分汉字
def hanzifreq(fcontent, sumdict): # 仅汉字词汇
    sent = " ".join(fcontent.splitlines()).strip().decode("utf8")
    sTuple=[ nltk.tag.str2tuple(t) for t in sent.split(" ") if hanzi.match(t) ]
    fredist=nltk.FreqDist(sTuple) #获取统计结果
    print len(fredist)
    for localkey in fredist:
        if localkey in sumdict : #检查当前词频是否在词典中存在
            sumdict[localkey]=sumdict[localkey]+fredist[localkey]
#如果存在, 将词频累加, 并更新词典值
        elif str(localkey[1]).find("None")==-1 :
            sumdict[localkey]=fredist[localkey] #将当前词频添加到词典中

sumdict={} # 统计结果
rootdir = " nltk_data/segments/" # 根路径
segpath = "swresult1998.txt" # 1998 年上半年 PFR 语料库的文件名
segcorpus = readfile(rootdir+segpath) # 读取语料为字符串格式
# fullfreq(segcorpus, sumdict) # 包含非汉字的词频统计
hanzifreq(segcorpus, sumdict) # 仅有汉字的词频统计

```

```

sumlist= sorted(sumdict.items(), key=lambda x:x[1],reverse=True)
file_str = StringIO()
for key in sumlist:
    file_str.write(str(key[0][0]));    file_str.write("\t")
    file_str.write(str(key[0][1]));    file_str.write("\t")
    file_str.write(str(key[1]));    file_str.write("\n")
savefile(rootdir+"freqdict.txt",file_str.getvalue()) #freqdict.txt
print "ok"

```

在分词语料的统计中还包括字频统计、词性统计等指标，这里就不一一列出，感兴趣的读者可以自己实现。

7.2.2 中文分词的测评

对一个中文分词系统的测评包括如下几个方面：分词正确率、切分速度、词典或语言模型的大小、功能完备性、易扩充性和可维护性 6 个方面。笔者认为后三个指标属于软件工程的范畴，而前三个才是考核分词系统性能的关键指标。

一般来讲，切分速度与具体的分词算法有关，目前无论使用 CRF 算法还是 ICTCLAS 的 NShort 最短路径算法，对于规模不大的分词任务，切分速度都能满足需求，也不属于性能的瓶颈。而如果执行超大规模的分词任务，较为常用的方法是使用现在流行的云计算平台（阿里云、新浪云、腾讯云等）。但从单机的性能上来看，ICTCLAS 查询词典可以使用双数组 Trie 树来存储和查询词典，性能得到较大提升。

从生成语言模型的大小来看，ICTCLAS 的二元关联词典远小于 CRF 算法生成的基于模板的语言模型。在实践中，特别是大规模、超大规模的词料分词，ICTCLAS 是首选。CRF 更多用于命名实体识别或领域自适应的专业分词系统。

在各项评测指标中，最核心的指标还是对切分精度的要求。传统上我们使用的切分精度为如下公式。

$$\text{总体测试分词正确率} \alpha = \frac{\text{切词结果中正确词次数}}{\text{评测语料中总词次数}} \times 100\%$$

前文所说的精度，都是通过这个公式得到的结果。除此之外，按照机器学习的精度统计，我们也提供了基于召回率和准确率的计算方法。这种方法常用于命名实体识别的评测中。

(1) 召回率 (Recall Ratio)。指系统在实施某单项测试时，识别出该测试项的能力。

设 K 为某单项测试项，召回率用公式表示为如下。

$$\text{召回率} R = \frac{\text{已识别出} K \text{的总数}}{\text{语料中} K \text{的总个数}} \times 100\%$$

(2) 精确率 (Precision Ratio)。指系统在实施某一单项测试项时，辨识出该项中正确成分的能力。设 K 为某单项测试项，精确率用公式表示为如下。

$$\text{精确率} R = \frac{\text{已识别出} K \text{的总数}}{\text{已识别项中} K \text{的总个数}} \times 100\%$$

例如，从新闻语料库中随机抽取了 1 000 个包含英译名的句子作为测试样本。这 1 000 个句子共含 6 107 个中文字符、1 537 个英语译名。系统运行后，辨识出“译名”2 376 个，其中有 1 507 个真正译名，则该系统的召回率和精确率分别如下。

$$\text{召回率} R = \frac{\text{已识别出} K \text{的总数}}{\text{语料中} K \text{的总个数}} \times 100\% = \frac{1507}{1537} \times 100\% = 98.05\%$$

$$\text{精确率} R = \frac{\text{已识别出} K \text{的总数}}{\text{已识别项中} K \text{的总个数}} \times 100\% = \frac{1507}{2376} \times 100\% = 63.43\%$$

该测试方法现在常用于各项 NLP 的序列标注任务、句法分析任务等多个方面，是较为全面衡量算法精度的通用指标。

7.2.3 宾州大学 CTB 简介

前面已经较为详细地介绍过宾州 CTB 汉语树库，读者对树库应用应该有了初步的了解。本节的侧重点是介绍宾州树库在使用中管理方面的方法。

宾州树库的语料代码为 LDC 2013T21，语料解压后，在 data 目录下有 4 个子目录，分别为：raw、segmented、postagged 和 bracketed。其中，raw 表示原始文本，segmented 是分好词的语料，postagged 为经过词性标注的语料，只有 bracketed 为手工标注的树库。需要说明的是，这 4 个目录中的语料可能由于版本更新的关系，并不是完全一一对应的，有些句子在 bracketed 树库中是两棵树，在其他的目录中却是一句话，但这种情况并不常见。为了减少不必要的核对的工作量，在实践中，可以统一从 bracketed 中抽取其他三种语料格式。

全部树库都保存在 bracketed 目录中，以 CTB 8.0 为例，bracketed（树库）目录内共有 3 007 个文件，可以从 ctb8.0-file-list.txt 文件中找到全部文件列表。这些文件共有 6 种不同的文件后缀，分别为：BC、BN、DF、MZ、NW 和 WB。每种后缀表示其保存的句

法树来自不同的来源。

- ❑ NW: Newswire: [0001-0325, 0400-0454, 0500-0540, 0600-0885, 0900-0931, 4000-4050]。
- ❑ MZ: Magazine articles: [0590-0596, 1001-1151]。
- ❑ BN: Broadcast news: [2000-3145, 4051-4111]。
- ❑ BC: Broadcast conversations: [4112-4197]。
- ❑ WB: Weblogs: [4198-4411]。
- ❑ DF: Discussion forums: [5000-5558]。

在 CTB 中, 多数保存句法树的文本文件都有如下两大类标签: 句法树专用 “()” 标签, 以及分隔不同句法树间的辅助标签, 诸如 “<>”。不同后缀的文件, 句法树专用标签虽然相同, 其保存句法树的辅助标签有细微差别。

在获得语料后, 应对源文件进行最初的解析, 目的是滤掉外部辅助标签, 从不同格式的文件中提取出句法树来, 然后保存到数据库中。实现代码如下。

(1) 数据库类: 需要安装 python-MySQL 库和 python 的 MySQLclient 库。

```
#!/usr/bin/python2.7
# -*- coding:utf-8 -*-

try:
    import MySQLdb
except ImportError:
    raise ImportError("[E]: MySQLdb module not found!")

class CMySQL(object):
    def __init__(self, host, pwd, user, db, port=3306):
        self.Option = {"host" : host, "password" : pwd, "username" : user,
            "database" : db, "port": port}
        self.__connection__()

    def __del__(self):
        if self.__conn: self.close()

    def __connection__(self):
        try:
            self.__conn = MySQLdb.connect( host = self.Option["host"], user =
self.Option["username"],
                passwd = self.Option["password"], db = self.Option["database"],
```

```

        port = self.Option["port"],charset='utf8')
    self.__dictcursor = MySQLdb.cursors.DictCursor
    except Exception, e:
        print e
        raise Exception("[E] Cannot connect to %s" % self.Option["host"])

def execute(self, sqlstate): # @todo: 增、删、改—不关闭数据库
    self.cursor = self.__conn.cursor()
    self.cursor.execute(sqlstate)
    self.commit() # 执行事物

def insert(self, sqlstate): # @todo: 增、删、改—不关闭数据库
    self.cursor = self.__conn.cursor()
    self.cursor.execute(sqlstate)
    lastinsertid = int(self.__conn.insert_id())
    self.commit() # 执行事物
    return lastinsertid

def query(self, sqlstate): # 查询全部记录
    self.cursor = self.__conn.cursor(self.__dictcursor)
    self.cursor.execute(sqlstate) # 查询
    return self.cursor.fetchall()

def querylist(self, sqlstate, size): # 查询多条记录
    self.cursor = self.__conn.cursor()
    self.cursor.execute(sqlstate) # 查询
    return self.cursor.fetchmany(size)

def queryone(self, sqlstate): # 查询一条记录
    self.cursor = self.__conn.cursor(self.__dictcursor)
    self.cursor.execute(sqlstate) # 查询
    return self.cursor.fetchone()

def close(self):
    self.__conn.close()

def commit(self):
    try:
        self.__conn.commit()
    except:
        self.__conn.rollback()
    raise

```

(2) 数据表及字段说明如下。

```
CREATE TABLE `penn_treebank` (
  `Id` int(11) NOT NULL AUTO_INCREMENT,
  `filename` varchar(255) DEFAULT NULL COMMENT '文件名',
  `seq` int(11) DEFAULT NULL,
  `sentence` text COMMENT '原始句子',
  `sentsegment` text COMMENT '分词后句子',
  `sentPOS` text COMMENT '词性标注后句子',
  `sentTree` text COMMENT '依存树',
  `flatTree` text COMMENT '展平的句法树',
  PRIMARY KEY (`Id`),
  KEY `sent` (`sentence`(10)),
  KEY `segment` (`sentsegment`(10)),
  KEY `file_name` (`filename`),
  KEY `file_seq` (`seq`)
) ENGINE=MyISAM AUTO_INCREMENT=71373 DEFAULT CHARSET=utf8 COMMENT='树库主表';
```

(3) 执行代码如下。

```
# -*- coding: utf-8 -*-
import sys, os
import re
import traceback
from framework import *
from treelib import *
from nltk.tree import Tree, ParentedTree
from MySQL import CMySQL
from MySQLdb import *

reload(sys)
sys.setdefaultencoding('utf-8')
rootdir = "treebanks/ctb8.0/data/bracketed/" #树库文件所在路径
DBconn = CMySQL("127.0.0.1", "root", "root", "treebanks", 3306)
filelist = os.listdir(rootdir)
count = 0
for filename in filelist:
    linelist = readfile(rootdir+filename).splitlines()
    linelen = len(linelist)
    treelist = []
    begin = end = -1; indx=0
    while indx < linelen: # 提取句法树字符串
        if (begin == -1) and linelist[indx] and linelist[indx][0]=="(":
```



```

        begin = indx
        elif (begin != -1) and (linelist[indx]=="" or linelist[indx][0]=="<"
or linelist[indx][0]=="(" or indx==(linelen-1)):
            end = indx
            if indx==(linelen-1) and linelist[indx] and linelist[indx][0]!="(":
                treelist.append("\n".join(linelist[begin:]))
            else:
                treelist.append("\n".join(linelist[begin:end]))
            if linelist[indx] and linelist[indx][0]=="(" : indx -= 1
            begin = end = -1
        indx +=1
file_seq = 0;
for treestr in treelist : #生成扁平树、句子字符串、句子分词、句子词性标注
    fieldlist = []
    sentTree = escape_string(treestr)
    flatTree = escape_string(treestr.replace("\n"," ").replace("\t"," "))
    try:
        mytree = Tree.fromstring(treestr)
        wordlist = mytree.leaves()
        sentSegment = escape_string(" ".join(wordlist))
        sentence = escape_string("".join(wordlist))
        sentPOS = escape_string(" ".join([word_pos[0][0]+"/"+word_pos[0][1]
for word_pos in flatten_deeptree(mytree).pos()]))
    except Exception,te: # 异常处理
        print filename,flatTree
        print te
        sys.exit()
    file_seq +=1
    fieldlist.append(escape_string(filename));
fieldlist.append(str(file_seq));
    fieldlist.append(sentence); fieldlist.append(sentSegment); fieldlist.
append(sentPOS)
    fieldlist.append(sentTree); fieldlist.append(flatTree);
    insertsql = "INSERT INTO penn_treebank VALUES('','+',',','".join
(fieldlist)+'') # 插入数据库的语句
    count +=1
    DBconn.insert(insertsql)
print count,"ok"

```

在解析过程中,不同版本的 CTB 可能会出现报出异常的情况,此时需要手工调整一下出错的文件。异常的情况不会很多,笔者在执行时遇到的错误大约有两处,手工修改之后即完成插入。

执行结果如图 7.7 所示。

Id	filename	seq	sentence	sentsegment	sentPOS	sentTree	flatTree
1	chtb_0001.nw	1	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
2	chtb_0001.nw	2	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
3	chtb_0001.nw	3	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
4	chtb_0001.nw	4	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
5	chtb_0001.nw	5	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
6	chtb_0001.nw	6	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
7	chtb_0001.nw	7	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>
8	chtb_0001.nw	8	<MEMO>	<MEMO>	<MEMO>	<MEMO>	<MEMO>

文本

十六进制编辑器

(((IP-HLN (NP-SBJ (NP-PN (NR 上海) (NR 浦东)) (NP (NN 开发) (CC 与) (NN 法制) (NN 建设))) (VP (VV 同步))))

图 7.7 CTB 数据库界面

CTB 数据表字段说明如表 7.2 所示。

表 7.2 CTB数据表字段说明

字 段 名	字 段 值
文件名 (filename)	chtb_0001.nw
在当前文件中的序列 (file_seq)	1
原始语料 (sentence)	上海浦东开发与法制建设同步
分词后句子 (sentSegment)	上海 浦东 开发 与 法制 建设 同步
词性标注后句子 (sentSegment)	上海/NR 浦东/NR 开发/NN 与/CC 法制/NN 建设/NN 同步/VV
句法树 (sentTree)	(((IP-HLN (NP-SBJ (NP-PN (NR 上海) (NR 浦东)) (NP (NN 开发) (CC 与) (NN 法制) (NN 建设))) (VP (VV 同步))))
扁平后句法树 (flatTree)	(((IP-HLN (NP-SBJ (NP-PN (NR 上海) (NR 浦东)) (NP (NN 开发) (CC 与) (NN 法制) (NN 建设))) (VP (VV 同步))))

全部语料插入数据库后，共生成 71 372 条记录。

7.3 语义知识库

语义知识库是现代语义网和百科知识库的前身,早期的语义知识库都是由手工创建的,虽然其规模远小于现在的百科知识库,但是经历了义原抽取、分类体系的规划、属性和属性值的界定几个步骤。它是现代语义理论在实践应用的产物。知识库的建设使用了义素分析法、本体论等语义学甚至哲学的理论。从小规模语料的实验结果来看,这类知识库基本可以满足语义辨析的需要。后来,W3C 组织公布的语义网(Semantic Web)规范,从理论上来看,两者之间一脉相承,差异不大。其中,HowNet 中的实体与 OWL 本体的概念相对应,事件与 Property 相对应,属性与 Attribute 相对应。唯一不同的是,语义网将知识库的构建从领域知识扩展到了整个互联网。

因此,通过对 HowNet 的深入研究,读者可以了解到如何构建一个汉语通用知识库的全过程,有助于读者从一个整体上把握一个知识库的完整构建原理和步骤。

7.3.1 知识库与 HowNet 简介

为了实现语义的辨析和挖掘,人们开发和建设了一些重要知识库。下面简要介绍早期的几个著名的语义知识库。在国际上最为知名的以词汇为基础的知识库为 WordNet,它是由普林斯顿大学认知科学实验室 George A. Miller 等开发的英语词汇知识库。WordNet 试图建立一个模仿人脑词汇组织原则的词汇网络,在构建中利用心理学的发现和心理词典的研究成果。在汉语领域,最为著名的是知网(英文名称为 HowNet),是一个以汉语和英语的词语所代表的概念为描述对象,以揭示概念与概念之间,以及概念所具有的属性之间的关系为基本内容的常识知识库。

HowNet 由中国科学院董振东先生开发,第一版于 1999 年发布。早在 1988 年前后,董振东就曾在他的几篇文章中提出有关知识库构建的观点。一开始,构建知网的目的是非常明确的,就是建立一个常识性的知识体系或语言以外的知识体系。1988 年董振东提出:

知识被假定为一个系统,它包含一个基于关系的和基于推理的系统。在基于关系的系统中有两种类型的关系:一种是概念特征之间的关系;另一种是事件之间的关系。概念由两部分组成,即概念核心和与概念相关的特征。推理系统是基于基关系工作的。有两种类型的推理:一种是基于概念特征的关系,其中典型的表现是替代;另一种是基于事件的关系,这种关系可以典型地看作"who-done-it"类的事件。

HowNet 是一个典型的语义场论的知识库体系。下面来研究一下其构建的一些细节内容。下文所述的内容部分参考了董振东先生的 *HowNet And the Computation of Meaning*。虽然目前计算语言学界已经多用 word2Vec 计算出的语义向量来表示词汇,并计算词汇之间的语义相似度,但是经过深入的研究,HowNet 的设计思想仍旧有精妙之处,值得学习。HowNet 对后代知识库系统的构建具有很重要的意义。

7.3.2 发掘义原

知网(HowNet)认为:“在计算隐含在文字和语言结构背后的意义时,应该确定一组意义的最小单位。而这些最小单位的意义是不能再被分解的。”这就是 HowNet 中义原的定义。显然,义原的概念也就是前文所说的义素。HowNet 还给出如下几个例子:“劫持”的含义可以被分解为“抢取”和“使用武力”,以及“行人”的含义可以被分解成“人”和“用脚行走”,而“抢取”、“人”、“行走”和“使用武力”似乎不能被进一步分解或者至少不容易再进一步分解。它们都作为知识库最基本的义原存在。

义素分析的理论是基于还原论思想的。这样就产生了如何才能找到汉语中的全部义原问题。经过多年的探索,人们发现,中文语言在字面水平就表现出相当多的语义信息。中国方式的汉字组成和概念形式都表明它们可以包含一种复杂的和语用的系统关系。

首先从造字法的角度来分析,汉语常用字有 6 000 多个字符,而大多数中文字都是形声字,由两部分构成:表义部分和表音部分。大多数字符由意义部分揭示了意义或给出相关意义的提示,而语音成分表示该字符的可能发音,字符“洋”由两部分组成。它的第一部分“氵”(称为“三点水”)是义部,与“水”有关;第二部分“羊”表示字符的发音。有趣的是,如下所有中文字都具有相同的义部“氵”:“江”、“湖”、“溪”、“海”、“流”、“泥”、“漕”,因为它们含义都与“水”有关。

如下是关于义部和声部更多的例子。

妈=(义部)女+(声部)马

房=(义部)户+(声部)方

吐=(义部)口+(声部)土

柏=(义部)木+(声部)白

睛=(义部)目+(声部)青

所有的中国文字都是由偏旁部首构成的,有时候偏旁本身就是一个字符。同时,几

乎所有的中文词和表达式都由文字组成，并且有时字符本身就可以被视为一个字。表 7.3 显示了由偏旁部首构成的汉字，在这里可以清楚地看到它是如何携带意义的。

表 7.3 偏旁部首与汉字

偏旁部首	含 义	汉 字
“氵”	水	河、海、溪、流
“钅”	金属	银、铁、铜、锈
“火”	火	烟、烤、烧、烫
“灬”	烧	热、煮、煎、照
“饣”	食物	饼、饿、饱、饮
...

一些中国字可作为单字词，如“走”、“医”，但有些不能，它们只能作为复音词和短语的组成部分。在现实中更多的是复音词和短语，换句话说，中国大部分的词语是多字符组合。同样的中文字和词句的组合更是语义相关的。因此，可以找到一些词汇，甚至短语的中文语言系统，这与印欧语言的区别是非常明显的。再从词汇的角度来看看如下两组，并研究中文方式的概念形式，不用在意它们是单词还是短语，虽然在传统的词典中它们都被视为单词。

如表 7.4 所示，在第一组中，所有的中文词汇都包含汉字“医”，意思是“给……治病”，在第二组中，所有的词汇都包含汉字“院”，其意义是“机构”。这是典型的中文构词方式。对于发现义原而言，最重要的是要找出表达一个词或短语的意义构成。很明显，它们都由两个有意义的组成部分构成。可以相信，义原一定存在于其中之一。在上述事实的启发之下，1989 年董氏父子开始了一项实验，他们选择了约 4 000 个常用的中国字，提取并列出它们所有的词义。他们使用了一个英文单词加一个中文词或表达式的三部分组合，用“|”进行分隔，并在义原中表示出意义。这里为了节省空间，知识库省略了中文部分。换句话说，在表 7.5 中像“abandon|”这样的符号不再是一个英文单词，而是表达为义项或义原。然后做了一个 4 000 个字符的义项表，在表 7.5 中列出。

表 7.4 复音词中的义项

词 汇	第一个汉字	第二个汉字	词 汇	第一个汉字	第二个汉字
医务	医 (Treat)	务 (affairs)	学院	学 (study)	院 (institution)
医师	医 (Treat)	师 (professional)	剧院	剧 (drama)	院 (institution)
医院	医 (Treat)	院 (institution)	法院	法 (law)	院 (institution)
医术	医 (Treat)	术 (skill)	画院	画 (painting)	院 (institution)
.....

表 7.5 语原列表

词	义 原
放	abandon add lend shoot
丢	abandon lose throw
置	buy put
借	borrow lend use
.....

如表 7.5 所示，通过删除所有的重复词义，抽取出初步测试的义原表，它包括 1 500 种不同的义项。虽然它覆盖了 4 000 个字符，但是类似的义项之间存在一些细微的差别。它需要一些细微的修改或调整。之后，又花费了 3 年的时间，使用这最初的 1 500 个义原为 50 000 个中文词和短语进行标记，并对应出含义相同的英语词。这个过程本身是一个大型语言工程实验，它虽然耗费了大量的劳动和时间，但证明它用于 HowNet 的后续研究和发展是非常有价值的。基于表 7.5 的一些义原，看看它们是如何支配概念的。其用于标记英文和中文词和短语。

最后，给出使用“eat|”和“use|”作为义原的两个例子，内容如下。

(1) “eat|”被提取为“to take solid food”的义原，它可以被用来标记如下。

英语：eat, take, take in, etc。

中文：吃、餐、用、食、进等。

(2) “use|”被提取为“to make work”的义原，它可以被用来标记如下。

英语：use, utilize, utilise, apply, employ, etc。

中文：用、动、使、采用、利用、应用、运用等。

7.3.3 语义角色

语义角色被定义为知网的内在关系，其中事件的参与者有真实的或想象的上下文语境，这也被称为主题角色或深层格。知网的语义角色来自超过 95 000 个中文义项、英语单词及固有语义角色的表达方式。在讨论知网的语义角色之前，有如下两点需要强调。

- (1) 语义角色不应该考虑语法结构的定义。
- (2) 它们是严格规定的，并能匹配每个事件类，而且每个事件都同时用中文和英语编写。

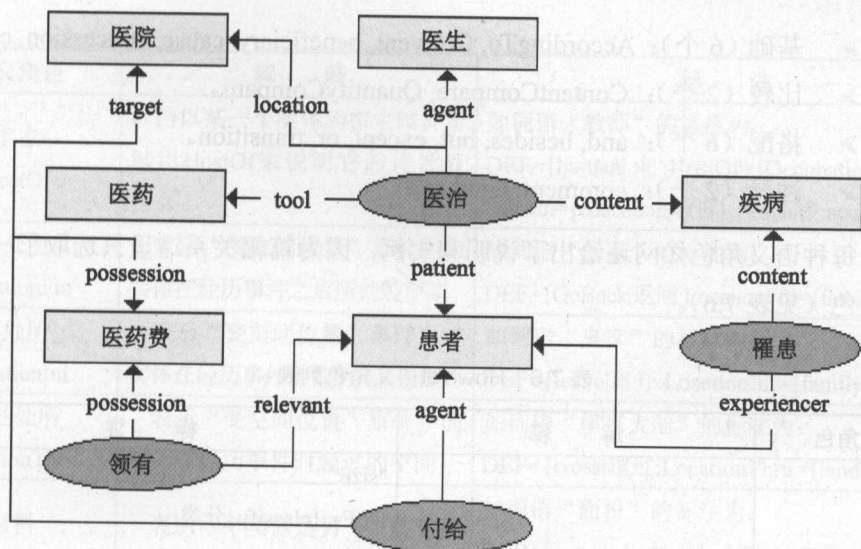


图 7.8 HowNet 语义角色图

如图 7.8 所示，带箭头的线段上的标签就是语义角色。知网使用了 91 个语义角色其分为两类：主要语义角色和周边语义角色。其子分类如下。

□ 主要语义分类角色。

- 基本语义角色 (6 个): agent, coagent, existent, experiencer, possessor, relevant.
- 受影响的语义角色 (11 个): isa, ContentProduct, PartOfTouch, PatientPart, atientProduct, content, contrast, partner, patient, possession, target.

□ 周边语义角色。

- 时间 (12 个): DurationAfterEvent, DurationBeforeEvent, EventProcess, SincePeriod, SincePoint, TimeAfter, TimeBefore, TimeFin, TimeIni, TimeRange, duration, time.
- 空间 (10 个): LocationFin, LocationIni, LocationThru, SourceWhole, StateIni, StateFin, direction, distance, location, source.
- 结果 (8 个): ResultContent, ResultEvent, ResultIsa, ResultWhole, Purpose, result, sequence, succeeding.
- 方式 (11 个): accompaniment, aspect, cost, degree, frequency, instrument, manner, material, means, method, times.
- 修饰语 (16 个): HostOf, MaterialOf, OfPart, PatientAttribute, PatientValue, RelateTo, Belong, concerning, descriptive, host, modifier, quantity, range, restrictive, scope, whole.

- 基础（6个）：AccordingTo, CoEvent, beneficiary, cause, concession, condition
- 比较（2个）：ContentCompare, QuantityCompare。
- 搭配（6个）：and, besides, but, except, or, transition。
- 注释（2个）：comment, emphasis。

对于每种语义角色知网还给出了说明和实例，因为篇幅关系这里只选取了一些主要的语义角色（见表 7.6）。

表 7.6 HowNet语义角色列表

语义角色	解 释	标 注
根据 AccordingTo	基于事件的实体	"size" DEF= {classify 分类: AccordingTo={Size 尺 寸 ".host={physical 物 质}}} "customization" DEF={produce 制造: AccordingTo= {aspiration 意愿}}
共享事件 CoEvent	说明一个实体类的概念与一个事件享有完全相同的角色框架。 例如，“信心”虽然是一个实体类的概念，但它属于“情感”类，它与“相信”这个事件类的概念共有相同的角色框架。“相信”的框架包括：experiencer、target和cause。而“信心”实际也含有这个相同的角色框架	"belief" 共享了"believe"同一角色框架：经验者和目标。 "wish" DEF= {aspiration 意 愿 :CoEvent= {expect 期 望}} "environmental protection" DEF={fact 事 情 :CoEvent={protect 保 护 : patient={Environment 情 况 :host={entity 实 体}}}}
内容作品 ContentProduct	表示“编辑”、“写”、“画”等类事件中被造就的实体	"publishing house" DEF={InstitutePlace 场 所 :{publish 出 版 : ContentProduct= {publications 书 刊 },agent= {~}}}
后延时段 DurationAfterEvent	事件停止后到说话者说话的时间段	如词语“久别”的标注为： DEF={farewell 离 别 : DurationAfterEvent = {TimeLong 长时间}}
事件过程 EventProcess	事件发生的伴随过程	如词语“疗程”的标注为： DEF={process 过程:{doctor 医治:EventProcess= {~}}}

续表

语义角色	解 释	标 注
宿主 HostOf	当以某一个实体为宿主时, 可以用HostOf来说明它的属性有什么	如词语“教师”的标注为: DEF={human 人:HostOf={Occupation 职位}, domain={education 教育},{teach 教:agent={~}}}
终处所 LocationFin	表示“变空间位置”事件中的实体在经历事件之后所处的空间	如词语“回家”的标注为: DEF={GoBack 返回:LocationFin={family 家庭}}
原处所 LocationIni	表示“变空间位置”事件中的实体在经历事件之前所处的空间	如词语“离家”的标注为: DEF={leave 离开:LocationIni={family 家庭}}
通过处所 LocationThru	表示“变空间位置”事件中的实体在经历事件时经过的空间	如词语“横贯大陆”的标注为: DEF={cross 越过:LocationThru={land 陆地}}
材料 MaterialOf	表明一个实体是另一个实体的材料	如词语“面粉”的标注为: DEF={material 材料:MaterialOf={edible 食物}, material={crop 庄稼}}
部分 OfPart	表示“蕴涵关系”或表示状态事件中“残疾”类事件中的实体的部件	如词语“插画”的标注为: DEF={image 图像:{contain 包含:OfPart={~}, whole={publications 书刊}}}
触及部件 PartOfTouch	表示“变形状”类事件中被触及的部件	如词语“击掌”的标注为: DEF={beat 打:PartOfTouch={part 部件:whole={ part 部件:PartPosition={hand 手},whole={ human 人}}}}
受事属性 PatientAttribute	表示“变属性”类事件中被变化的实体新获得的属性值, 通常它已经蕴含在词语内部	如词语“调色”的标注为: DEF={adjust 调整:PatientAttribute={Color 颜色}}
受事部件 PatientPart	表示“部件他移”类事件中被移动的实体的部件	如词语“躬身”的标注为: DEF={CausePartMove 部件他移:PatientPart={ part 部件:PartPosition={body 身},whole={ human 人}}}
受事属性值 PatientValue	表示在“变属性”类概念中被改变的属性所获得的具体的属性值	如词语“加高”的标注为: DEF={AlterForm 变形状:PatientValue={high 高}}
相关RelateTo	表示一个概念与什么样的信息有关, 但这种关系可能是不清晰的	如词语“婚纱”的标注为: DEF={clothing 衣物:RelateTo={GetMarried 结婚}}
结果内容 ResultContent	表示“处罚”等类事件中的结果	如词语“罚金”的标注为: DEF={expenditure 费用:{punish 处罚:Result Content={~}}}

续表

语义角色	解 释	标 注
结果事件 ResultEvent	表示“使之动”等事件所导致的、由其受事施行的事件	如词语“打伤”的标注为: DEF={beat 打:ResultEvent={wounded 受伤}}
结果类指 ResultIsa	表示“使之是”或“使之非”等事件被类指的实体	如词语“解职”的标注为: DEF={dismiss 罢免:ResultIsa={Occupation 职位}}
结果整体 ResultWhole	表示“纳入”或“分类”等事件中的实体的整体	如词语“入伙”的标注为: DEF={include 纳入:ResultWhole={community 团体}}
来源整体 SourceWhole	表示“变包含”等事件中的作为来源的整体	如词语“退伙”的标注为: DEF={withdraw 退出:SourceWhole={community 团体}}
终状态StateFin	实体在经历事件之后的状态	如词语“复旧”的标注为: DEF={resume 恢复:StateFin={past 过去}}
原状态StateIni	实体在经历事件之前的状态	如词语“恢复知觉”的标注为: DEF={BeRecovered 复原:StateIni={dizzy 昏迷}}
之后TimeAfter	表示一个事件发生是在另外一个事件发生之后	如词语“圈阅”的标注为: DEF={sign 题写:TimeAfter={read 读}}
之前 TimeBefore	表示一个事件的发生是在另外一个事件发生之前	如词语“饭前”的标注为: DEF={time 时间:TimeBefore={eat 吃}}
伴随 accompaniment	两个以上的事件形成的伴随关系	如词语“风险投资”的标注为: DEF={provide 供:accompaniment={dangerous 危},domain={economy 经济},possession={fund 资金}}
施事agent	表示行动的事件类型中“变关系”、“变状态”、“变属性”、“使之动”四类事件中的充当“变”这一功能的实体	如词语“航行”的标注为: DEF={VehicleGo 驶:agent={ship 船}}
归属belong	表示某个成员属于一个整体,但是这个整体不会因为这个成员的多少或者存亡而发生变化	如词语“汉字”的标注为: DEF={character 文字:belong="China 中国"}
受益者beneficiary	事件中的受益的实体	如词语“为人民服务”的标注: DEF={do 做:beneficiary={human 人}:quantity={mass 众}}

续表

语义角色	解 释	标 注
原因cause	事件发生的原因	如词语“死因”的标注为: DEF={cause 原因:{die 死:cause={~}}}
关于concerning	表示涉及某实体	如词语“动物学”的标注为: DEF={knowledge 知识:concerning={animal 兽}}
让步concession	事件发生或进行的让步	如词语“徒有其表”的标注为: DEF={useless 无用:concession={GoodLooking 好看}}
条件condition	事件发生或进行的条件	如词语“买一送一”的标注为: DEF={obtain 得到:condition={buy 买},manner={FreeOfCharge 免费}}
内容content	表示“浮动”、“精神状态”、“变精神”等类事件中涉及的实体,它与“受事”的区别在于:“内容”不承受“改变”	如词语“好学”的标注为: DEF={FondOf 喜欢:content={study 学习}}
参照体contrast	表示“相比关系”、“变相比”、“相适关系”或“获胜”、“强过”等事件中的被参照的实体,即回答在比较事件中“跟谁比”或“跟什么比”	如词语“室外”的标注为: DEF={location 位置:contrast={room 房间},modifier={external 外}}
代价cost	表示“变领属”事件中为改变占有物的领属关系所付出或取得的实体	如词语“邮资”的标注为: DEF={expenditure 费用:{post 邮寄:cost={~}}}
程度degree	事件或属性值的程度	如词语“酷似”的标注为: DEF={BeSimilar 相像:degree={extreme 极}}
描写体descriptive	表示“是非关系”类事件中的用以描写主体的属性值	如词语“变红”的标注为: DEF={become 成为:descriptive={red 红}}
方向direction	表示“时空关系”和“变空间位置”等事件中的实体在经历事件时其所处的或其位移的方向	如词语“南迁”的标注为: DEF={AlterLocation 变空间位置:direction={south 南}}
距离distance	表示距离的远近	如词语“远赴”的标注为: DEF={LeaveFor 前往:distance={far 远}}
进程时段duration	事件进行的从起始到终止的时间	如词语“昙花一现”的标注为: DEF={exist 存在:duration={TimeShort 短时间}}

续表

语义角色	解 释	标 注
存现体existent	表示关系的事件类型中的“时空关系”类事件的主体，以及表示“存现”状态类事件的主体	如词语“生病”的标注为： DEF={happen 发生:existent={disease 疾病}}
经验者experienter	表示状态的事件类型中除表示“存现”状态外的各类事件的主体	如词语“获胜者”的标注为： DEF={human 人:{win 获胜:experienter={~}}}
频率frequency	事件发生或进行的频率	如词语“重建”的标注为： DEF={build 建造:frequency={again 再}}
宿主host	属性的主人即宿主	如词语“颜色”的标注为： DEF={Color 颜色:host={physical 物质}}
工具instrument	事件发生或进行所依赖的工具	如词语“通信”的标注为： DEF={communicate 交流:instrument={letter 信件}}
类指isa	表示“是非关系”类事件中的被类指的实体	如词语“成俗”的标注为： DEF={become 成为:isa={Habit 习惯}}
处所location	事件发生的空间	如词语“接待站”的标注为： DEF={InstitutePlace 场所:{entertain 招待:location={~}}}
方式manner	事件发生或进行的方式	如词语“欢呼”的标注为： DEF={cry 喊:manner={joyful 喜悦}}
材料material	事件发生或进行所依赖的材料	如词语“鱼片”的标注为： DEF={food 食品:material={fish 鱼}}
手段 means	事件发生或进行所依赖的手段	如词语“换得”的标注为： DEF={obtain 得到:means={exchange 交换}}
方法method	表示事件进行所采用的方法	如词语“茶道”的标注为： DEF={method 方法:{drink 喝:method={~}}}
描述modifier	为被修饰对象增加某种属性值	如词语“白云”的标注为： DEF={CloudMist 云雾:modifier={white 白}}
相伴体partner	表示“关联关系”、“变关联”、“时空关系”或“较量”等事件中的与主体相对的实体	如词语“斗牛”的标注为： DEF={compete 比赛:partner={livestock 牲畜}}
受事patient	表示行动的事件类型中“变关系”、“变状态”、“变属性”、“使-动”四类事件中的充当“被改变”这一功能的实体	如词语“捕鱼”的标注为： DEF={catch 捉住:patient={fish 鱼}}

续表

语义角色	解 释	标 注
占有物possession	表示“领属关系”及“变领属关系”类事件中的被领属者	如词语“弃权”的标注为: DEF={abandon 放弃:possession={rights 权利}}
领有者 possessor	领属关系的占有者	如词语“受益者”的标注为: DEF={human 人:{obtain 得到:possession={Advantage 利},possessor={~}}}
目的purpose	事件发生或进行的目的	如词语“借阅”的标注为: DEF={borrow 借入:purpose={read 读}}
数量quantity	为被修饰对象增加数量值	如词语“千家万户”的标注为: DEF={family 家庭:quantity={many 多}}
幅度range	事件发生或进行的幅度	如词语“博览”的标注为: DEF={read 读:range={extensive 泛}}
关系主体relevant	表示关系的事件类型中除表示“领属关系”和“时空关系”以外的各类事件的主体	如词语“同义词”的标注为: DEF={expression 词语:{BeSame 相同:relevant={~},scope={information 信息}}}
限定restrictive	对被修饰对象的限定而非增加某种属性值	如词语“废气”的标注为: DEF={gas 气:restrictive={waste 废物}}
结果result	事件所导致或产生的结果	如词语“填满”的标注为: DEF={fill 填入:result={full 满}}
范围scope	事件发生或进行的范围或涉及的方面	如词语“贬值”的标注为: DEF={BecomeLess 减少:scope={Worth 价值}}
次序sequence	表示事件在进行中所表现出的顺序或者事件产生的次序	如词语“初露”的标注为: DEF={CauseToAppear 显现:sequence={first 首次}}
来源source	作为占有物包括物质的或精神的实体的来源	如词语“水井”的标注为: DEF={facilities 设施:{take 取:possession={water 水},source={~}}}
接续succeeding	两个以上的事件接连发生或进行,同时它们是密切相关的	如词语“围剿”的标注为: DEF={surround 包围:domain={military 军},succeeding={destroy 消灭}}
目标target	事件所涉及的、但没有被“改变”的实体	如词语“楷模”的标注为: DEF={human 人:{imitate 模仿:target={~}}}
时间time	事件发生的时间	如词语“朝露”的标注为: DEF={RainSnow 雨雪:time={time 时间:TimeSect={morning 晨}}}

续表

语义角色	解 释	标 注
整体whole	表示“蕴涵关系”或表示行动中“选择”类事件中的实体的整体	如词语“扶手”的标注为: DEF={fittings 配件:whole={implement 器具}}

7.3.4 分类原则与事件分类

HowNet 的分类系统很有特点，它不是机械地按照静态实体（Things 或 Entity）和动态事件（Events）进行分类，而是将构成静态实体和动态事件的所有组件，包括属性（Attributes）、属性值（Attributes-Values）、时间（Time）、部分（Parts）和空间（Space）都作为顶层的分类根节点，可以看到即使在当时就已经渗透了面向对象的设计思想。图 7.9 反映了这些根节点之间的关系。

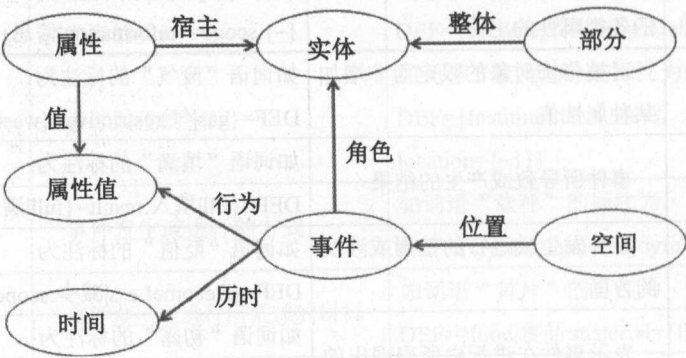


图 7.9 根节点之间的关系

如图 7.9 所示，实体（Things）是属性（Attributes）的宿主，也就是说属性依赖于实体，属性可以有不同的取值（Attributes-Values）。实体可以包含不同的部分（Parts），相对于部分，实体是整体。实体还是事件（Events）的角色。事件有发生的时间（Time）和所处的空间（Space），事件行为的结果可以导致属性值的改变。

同时，HowNet 在设计分类体系时，严格遵循如下原则。

（1）仅概念被考虑在内。被分类的是概念而不是词汇。例如，概念“植物”，也就是概念“树”的上位词，因为一个词可能是有歧义的，但概念则不同。当我们说“植物”是“树”的上位词时，“植物”应该是“活的有机体”，而“树”的概念应该是“一个高大的多年生植物”。

(2) 下位词的概念必须继承其上位词的概念所有的基本属性,同时,它必须至少有一个属性,是它的上位词的概念所不具备的。

(3) 分类方案不允许存在杂项或杂烩类。每当一个实体被发现从任何一类中脱离出来,那么整个分类方案都将是有瑕疵的,并需要重新设计分类方案。

(4) 任何类都可以被视为定义真实世界概念的义原。换句话说,HowNet 中义原的确定和概念的分类是同时进行的。

(5) 不同类型的概念,如事物、部分、事件、属性等,都应该有自己的唯一分类方案。

(6) 不允许类别之间的重叠是指,换句话说,任何一个概念只允许有一个类或一个直接的上位概念。当实在无法避免时,在 HowNet 中的原语可以弥补这样的情况。

HowNet 中的事件是指发生在一个特定的时间和空间的语言,在大多数情况下,由动词表示。在 HowNet 中有 812 个事件类。事件的概念可分为两种类型:内在的事件和外在的事件。内在的事件包括:(1)事物的发展(包括物理事物的发展和精神事物的发展)。(2)静态事件和动态事件之间的对应关系。(3)事件的蕴涵。(4)事件(通常由动词表示)与事实(由名词通常表示)之间的同源(RoleFrame)角色框架。外在的事件包括:(1)事件参与的事物。(2)在事件、属性和属性值之间的关系。(3)事件转换的语义角色。(4)主题域的特征。

该事件的概念可以细分为如下两个更高层次的类别组,如图 7.10 所示。

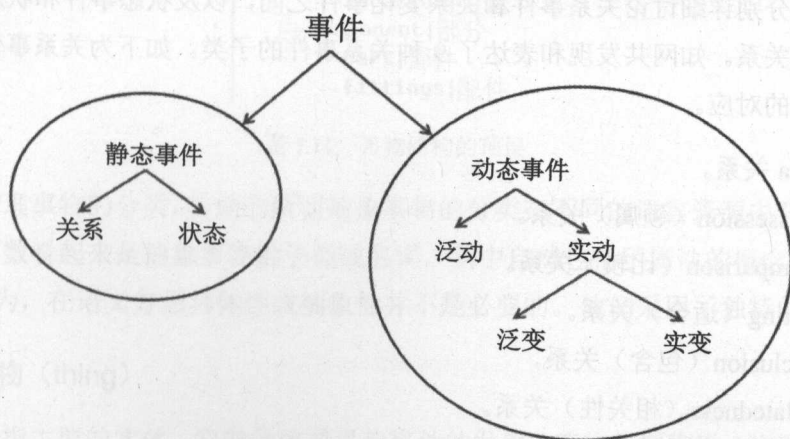


图 7.10 高层事件层次结构

第一组表示非现实的动作,而第二组指现实的动作。第一组划分成两个子组:关系

的事件和状态的事件。第二组指动作的事件，也细分为如下两个子类：一般动作，如“做”、“制作”等，以及具体的动作，如“吃”、“买”、“走”等。

一般认为，知网原创最显著的独特之处是双轴（Biaxial）事件理论。该理论基于 812 个事件类并标记了超过 20 000 个中文和英文动词。该理论揭示了事件的内在双轴方案。两轴彼此相交。一轴是事物发展，另一轴是静态事件与动态事件之间的对应关系。双轴理论被证明是由知网研究人员所做的重要的贡献之一。

（1）事物的发展：事物的基本特征反映了所发生的事物，即事物存在的状态和事物发生的变化。

虽然事物在形式和性质上有很大的不同，但它们都有共同的基本发展过程。物理的事物，无论是有生命的还是无生命的或社会单位，都会经历基本相同的发展过程，也就是形成，之后成长和成熟，逐渐变为老、弱，最终物理生命衰亡的发展过程。换句话说，它们经历了从生到死或从出现到消失的过程。对于任何发生的事情，如会议、生日聚会或公司，甚至政府的发展基本过程几乎相同，即开始或出现，可能会进入一个高潮或以后蓬勃发展，终于结束或消亡。

（2）静态事件和动态事件之间的对应。

知网经过研究已经发现静态事件概念和动态事件概念之间的完美对应。任何关系或状态都由其对应的动作引发或改变。例如，表示占有关系的事件，如“有”、“拥有”或“丢失”；对应于表示占有关系的变化，如事件“取”和“借用”或“给予”和“借出”。下面两节将分别详细讨论关系事件和关系变化事件之间，以及状态事件和状态变化事件之间的对应关系。知网共发现和表达了 9 种关系事件的子类。如下为关系事件和关系改变事件之间的对应。

- ☐ is-a 关系。
- ☐ possession（领属）关系。
- ☐ comparison（比较）关系。
- ☐ fitting（适合）关系。
- ☐ inclusion（包含）关系。
- ☐ relatedness（相关性）关系。
- ☐ cause-result（原因—结果）关系。
- ☐ location/time（位置/时间）关系。
- ☐ arithmetic（数学）关系。

显然，上述 9 类事件表示的既不是行为也不是状态。它们只是表达一些特殊种类的关系。例如，“他是一个教师，事件“是”表示 is-a 关系；在“我的哥哥有一辆汽车”的事件中，“有”表示领属关系，这一关系连接了“我的哥哥”和“车”。同样的，在“我的父亲是在伦敦”事件中，“是”表示位置关系，也就是，“我的父亲”和“伦敦”之间的关系。事件不表示“哥哥”和“汽车”或“父亲”和“伦敦”的任何状态，它们也不表达任何行为。上述 9 种类型的关系是由它们相应的关系变化的事件造成的。简而言之，实体之间的任何关系都是由对应该关系的行为引起的。例如，领属关系如 possess、own 或 do not have 是由表示领属改变的行为引起的，如 take、buy、borrow 和 give、sell、lend，等等。

7.3.5 实体分类

如图 7.11 所示，知网的实体分类包含的万物、部件、时间和空间是其顶级节点的义原。知网有 151 个实体分类，覆盖“万物”、“时间”、“空间”和“部件”。



图 7.11 万物结构的顶层

对比物理事物的分类，传统的所谓抽象事物的分类在不同的语言资源中有很大变化。其中的大多数看起来是抽象事物的子类或名词，其中自然包括了属性的概念（或名词）。HowNet 认为，在语义方面具体性或抽象性并不是必要的。他们采用了独特的方案。

1. 万物 (thing)

万物是指主要的实体，它能导致或引起事件的发生。万物包括物质（物理事物）、精神（心理事物）和群体（社会单元）。物理事物是由有物质形态的材料构成的对象，而心理事物没有物质形态。物理事物包括有生命的物质和无生命的物质。心理事物包括发生的心理特征。万物可被划分为如图 7.12 所示的形式。

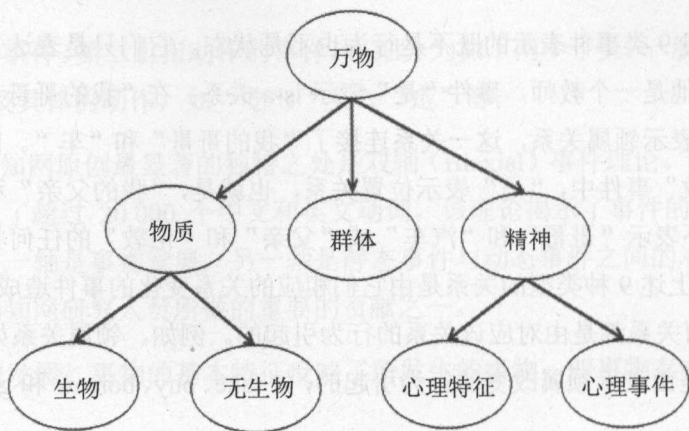


图 7.12 实体分类的架构

类似于知网事件的分类，实体分类不仅是实体的上位一下义关系的层次结构，而且也是义原定义的表示。从某种意义上说，它为公理和推理规则做准备。下面出详细的解释。

对实体分类的部分作出如下解释。

□ 一个“实体”的下义词是“万物”；“万物”的下义词是“物质”；“物质”的下义词是“生物”；“生物”的下义词是“动物”；“动物”的下义词是“人”；最后“拟人物”是“人”的下义词。这是上位一下义关系的层次结构，如图 7.12 所示。

□ 义原的定义传达了如下概念的详细信息。

- “万物”是“存在（在世界上）”，是“实体”。
- “物质”是一个“万物”，它有着“外观”的属性，可以“被感知”。
- “生物”是一种“物理事物”，里面有“年龄”的属性，可以“活着”，“死去”和“具有代谢过程”，可以“繁殖后代”及“被繁殖”。
- “包括人类的动物”，是一个“生物”，里面有“性别”的属性，可以“移动”，可以“体验心理状态（如感觉）”。
- “人”是具有某种特殊属性的“动物”，如“姓名”、“能力”和“智慧”，并能“思考”和“说话”。
- “拟人物”（如妖精）是一个“人”，但“不是真实的”。

□ 如前上位词属性的继承所述，知网指定任何概念都继承了上位词的所有基本属性，并且必须至少有一个属性是上位词所不具有的。这里，实体分类部分展示在图 7.12 中。“人”继承了所有上位词的属性。“人”不仅有“名字”、“能力”和“智慧”属性，还有“性别”（来自“动物”）、“年龄”（来自“生物”）和“外

观”(来自“物理事物”)属性。此外,“人”不仅能“思考”和“说话”,而且可以“移动”和“体验心理状态”(来自“动物”),并可以“活着”、“死去”,以及“新陈代谢”、“繁殖后代”和“被繁殖”(来自“生物”)。

- 上义词义位的覆盖和传递。
 - 根据实体的分类(一)。

{thing|万物} {entity|实体:{ExistAppear|存现:existent={~}}}

“thing”可以有“ExistAppear|存现”。事件的部分分类如图 7.13 所示。

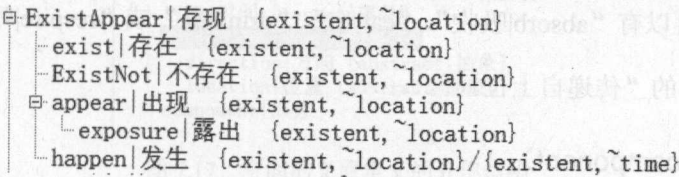


图 7.13 事件的部分分类

“thing”也可以有“exist|存在”、“ComeToWorld|问世”、“ExistNot|不存在”、“appear|出现”、“exposure|露出”或“happen|发生”。

- 根据实体的分类(二)。

{animate|生物} {physical|物质:HostOf={Age|年龄}, {alive|活着:experiencer={~}}, {die|死:experiencer={~}}, {metabolize|代谢:experiencer={~}}, {reproduce|生殖:agent={~}, PatientProduct={~}}}

“living thing”可以有“metabolize|代谢”(对于生命是必需的)。事件的部分分类如图 7.14 所示。

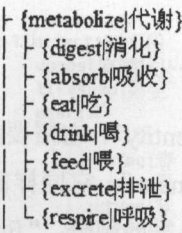


图 7.14 事件的部分分类

“living thing”可以有“digest|消化”、“absorb|吸收”、“eat|吃”、“drink|喝”、“feed|喂”、“excrete|排泄”或“respire|呼吸”。

这就是所说的覆盖下义词。

➤ 实体分类如图 7.15 所示。

实体分类如图 7.15 所示。

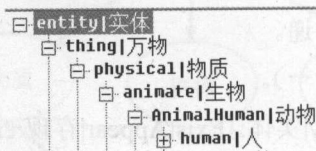


图 7.15 实体分类

“human”可以有“absorb|吸收”、“eat|吃”、“drink|喝”或“respire|呼吸”。

这就是所谓的“传递自上位词”。

2. 部分 (component)

在知网中，使用“component|部分”义原指向“part”的根节点。“component|部分”包含两个义原子类：“part|部件”和“fittings|配件”，如图 7.16 所示。

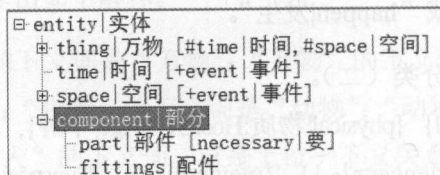


图 7.16 实体分类体系中的部分

实体部分分类的解释如下。

(1) “thing”和“component”是上位词，同为“entity”的兄弟节点，并且“component”有两个下义词：“part”和“fittings”。

(2) 义原的定义给出概念的进一步信息。

- “component”是一个“entity”，它需要结合另一个“entity”才能形成整体。
- “part”是一个“component”，它同样需要结合另一个“entity”才能形成整体。“part”还是其中必需的部分。“fittings”是一个“component”，它同样需要结合另一个“entity”才能形成整体，但并非是绝对的必要。例如，按照知网的定义，鞋子的鞋底是“part”，鞋子的鞋带是“fittings”。

3. 空间

其实很多事情隐含着空间的概念，如土地、水、天空和场所等，或机构，如学校、

医院、商店等，或建筑物，如房子等，甚至许多器物，如汽车、家具、容器等。“空间”作为 7 个根节点类之一，可以高度抽象地分为如下两个子类。

- (1) 方向，如东、西、风向等。
- (2) 地点，如地址、点、入口。

在实体分类中，空间的义原定义的分层结构如图 7.17 所示。

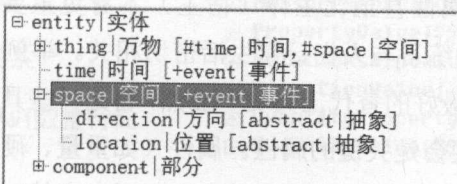


图 7.17 空间的义原定义的分层结构

4. 时间

类似于空间和场所，时间有其自身的特征，HowNet 称之为片段 (section)，不同于子类或部件，因为它们既不是子类，也没有部件。在知网中，它们被置于次要特征的分类中。

一个经常提出的问题是，为什么不把时间分为“时间点”和“时间段”？这个建议听起来很合理，有时可能是有用的，但实践起来是非常困难的。是否是一个时间点或一个时间段可能会在许多情况下依赖于有关事件的视图。例如，“下午”是时间段还是时间点呢？在我们看来，在许多情况下，它取决于与之相关的事件。

时间都有其特殊的片段和特性，在知网的二级特征列表中指定，如图 7.18 所示。



图 7.18 实体分类体系中的时间

7.3.6 属性与分类

1. 属性

知网中的属性为任何对象所必须拥有的一组属性。对象之间的相似性和差异是由它们携带的属性来确定的。没有无属性的对象。人类被附加了自然属性，如种族、肤色、性别、年龄、思考能力和语言运用；以及社会属性，如民族、家庭出身、职业、财富等。在特定条件下，附加的属性甚至比主体（宿主）本身更重要，事实最明显地体现在，“next-best alternative”的练习与人们日常生活密切相关。举例来说，我们要在墙上钉钉子，但没有锤子，什么是最好的替代工具呢？显然，将是一些具有接近锤子属性的东西，在此情况下，重量和硬度会是关键的属性。属性（如重量、硬度等）与宿主（锤子）之间的关系是直接的。这些属性只是来源于其宿主，反之亦然。属性—主体的关系不同于部分—整体的关系。知网通过编码的方式反映了这种差异。这样属性必须被定义到可能的宿主类中，代码如下。

```
"depth"
DEF= {Depth|深度:host={physical|物质}}

"depth"
DEF={Abstmsemss|深浅:host={Content|内容}}
```

事实上，自然语言属性的定义也表明了它们的宿主。在 WordNet 中“deep”的定义如下。

- (1) “deep”：程度向下、向后或向内。
- (2) “deep”：心理或智力深度的程度。

知网的属性分类包含 247 个义原类，并且它们被划分为 7 个子类，如图 7.19 所示。



图 7.19 属性分类

2. 属性值的分类

一方面，每个属性都指向其宿主，另一方面也指向其对应的值。换句话说，属性和

属性值之间存在内在的相互参照的关系。所有的属性都有其对应的值，同时所有属性值都有相应的属性。

知网的属性值分类包含 889 个义原类。正如知网之前所述，该属性被划分为 7 个子类，属性值可以固有地被分为 7 子类，对应于图 7.20 所示的属性子类别。

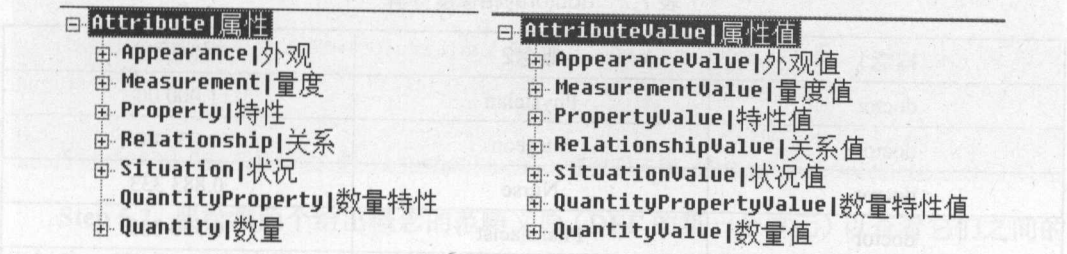


图 7.20 属性子类别

知网浏览器提供了属性和对应的属性值的详细列表，用于计算的关系，如图 7.21 和图 7.22 所示。

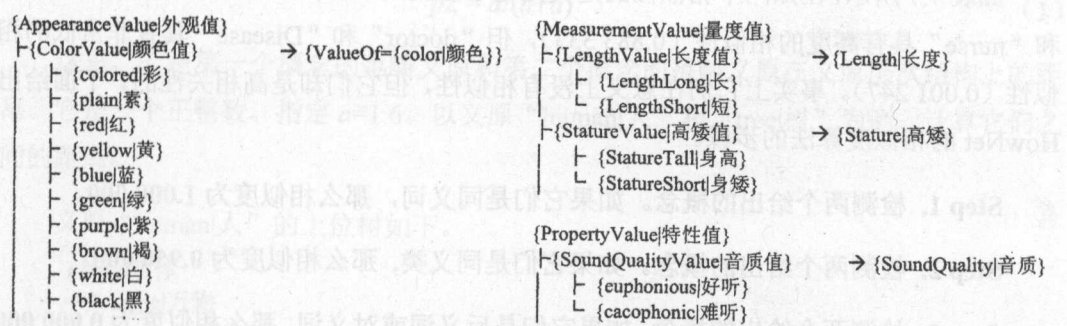


图 7.21 属性值列表（1）

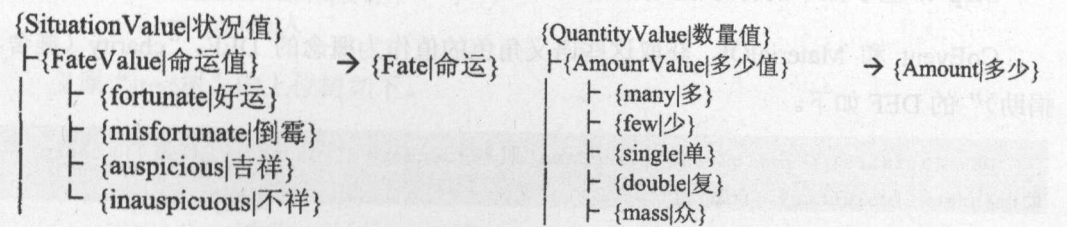


图 7.22 属性值列表（2）

7.3.7 相似度计算与实例

在知网中特别设计了一个模块可以采集和表示概念的相似度，名为概念相似性计算器（CRC）。此装置能够给出知网中任何一个中文或英文词汇的任何意义的概念场。

概念相似性度量 (CSM) 能够度量知网中任何词语和短语的任何词义之间的相似度。在知网中相似性的测量是基于概念的，它已被证明达到非常高的精确度。表 7.7 显示了概念 “doctor” 之间的相似性，表示为 “licensed medical practitioner”，并分别与医学领域的其他概念相关。

表 7.7 doctor的相似度计算

概念1	概念2	相似度值
doctor	Physician	1.000 00
doctor	Surgeon	0.950 00
doctor	Nurse	0.883 333
doctor	Pharmacist	0.550 000
doctor	Anesthetist	0.340 000
doctor	Patient	0.203 636
doctor	Disease	0.018 605

如表 7.7 所示，在知网中相似性的定义是基于意义的，与关联性无关。例如，“doctor” 和 “nurse” 具有高度的相似性 (0.883 333)，但 “doctor” 和 “Disease” 具有非常低的相似性 (0.001 247)。事实上它们在意义上没有相似性，但它们却是高相关性的。下面给出 HowNet 的相似度算法的步骤。

- Step 1. 检测两个给出的概念。如果它们是同义词，那么相似度为 1.000 000。
- Step 2. 检测两个给出的概念。如果它们是同义类，那么相似度为 0.950 000。
- Step 3. 检测两个给出的概念。如果它们是反义词或对义词，那么相似度为 0.000 000。
- Step 4. 基于如下的特殊语义角色进行计算。

CoEvent 和 MaterialOf，获取这些语义角色的值作为概念的 DEF。“charity（施舍、捐助）” 的 DEF 如下。

```
DEF={affairs| 事务 :CoEvent={TakeCare| 照料 :manner= {voluntary| 自愿}} {help| 帮助:manner= {voluntary | 自愿}}}
```

当计算 “charity” 的相似度时，使用它的 DEF，代码如下。

```
{TakeCare|照料:manner={voluntary|自愿}} {help|帮助:manner={voluntary|自愿}}
```

Step 5. 检测两个给出的概念。如果它们不适合前三步的情况，那么一些更加精细的计算会被执行。

Step 5.1. 检测两个给出的概念，看看它们的 DEF 是否是相互包含的，p1 的值将被

给出。如果两个 DEF 是相互包含的, 那么 $p1=1.000\ 000$; 如果不是, 那么 $p1=0.000\ 000$ 。
“psychotherapist” (心理医生) 和 “doctor” (医生) 的 DEF 如下。

```
" psychotherapist ":
  DEF={human| 人 :HostOf={Occupation| 职位 },domain={medical| 医 },{doctor| 医治 :
agent={~},content={disease|疾病:cause={mental|精神} }}}
```

```
" doctor ":
  DEF={human| 人 :HostOf={Occupation| 职位 },domain={medical| 医 },{doctor| 医治 :
agent={~}}}
```

“doctor”的 DEF 能够被嵌入到 “psychotherapist”的 DEF 中, 那么 $p1 = 1.000000$ 。

Step 5.2. 要检测两个给出概念的范畴义原 (DEF 的第一个单元) 以查看它们之间的相似度, 那么 $p2$ 将被给出。

当所有的义原被组织进一个层次结构时, 表示为上下位的关系。假设两个义原的距离是 d , $p2$ 可由公式 (1) 计算:

$$p2 = a/(d+a) \quad (1)$$

这里, d 是第一个概念的范畴义原和第二个概念的范畴义原在义原层次结构上的距离。它是一个正整数。指定 $a=1.6$ 。以义原 “human|人” 和 “tree|树” 为例, 计算它们之间的距离。

义原 “human|人” 的上位树如下。

```
entity|实体
==> thing|万物
    ==> physical|物质
        ==> animate|生物
            ==> AnimalHuman|动物
                ==> human|人
```

义原 “tree|树” 的上位树如下。

```
entity|实体
==> thing|万物
    ==> physical|物质
        ==> animate|生物
            ==> plant|植物
                ==> tree|树
```

这两个义原共同的上位词为 “animate|生物”。这两个义原到它们共同的上位词的长度各自为两步, 那么这两个义原之间的距离为 4 步, 也就是 $d=4$ 。

$$p2 = a/(d+a) = 1.6/(4+1.6) = 1.6/5.6 = 0.285714.$$

Step 5.3. 检测在描述节点内的两个概念之间的相似度, p_3 将被给出, 基于公式(2)进行计算。

$$p_3 = N_s * 2 / (N_{c1} + N_{c2}) \quad (2)$$

注意: 描述节点由 "semantic role = sememe" 或 "ZeroRole = sememe"组合构成。

其中, N_s 表示给出的两个概念的 DEF 中同一描述节点的数量; N_{c1} 表示给出的第一个概念的 DEF 中描述节点的数量; N_{c2} 表示给出的第一个概念的 DEF 中描述节点的数量。比较如下两个概念的描述节点, 并将比较结果展示在表 7.8 中。

```
" private instructor " (家庭教师、私教):  
DEF={human|人:{teach|教:agent={~} ,location={family|家庭}}}  
" assistant" (助理):  
DEF={human|人:HostOf={Occupation|职位},{teach|教:agent={~}}}
```

表 7.8 描述节点之间的比较结果 (一)

"private instructor"的描述节点	"assistant"的描述节点
<i>ZeroRole={human 人}</i>	<i>ZeroRole={human 人}</i>
	<i>HostOf={Occupation 职位}</i>
<i>ZeroRole={teach 教}</i>	<i>ZeroRole={teach 教}</i>
<i>Agent={~}</i>	<i>Agent={~}</i>
<i>Location={family 家庭}</i>	

根据表 7.8 可知, "private instructor"和"assistant"有三个相同的描述节点(in italic)。在这个阶段相似度被计算为如下结果。

$$p_3(\text{private instructor}, \text{assistant}) = 3 * 2 / (4 + 4) = 6 / 8 = 0.750000.$$

Step 5.4. 在知网实体分类层次结构中, 每个语义类都有一个定义。

"human|人"有如下定义。

```
DEF={AnimalHuman|动物:HostOf={ Ability |能力} {Name|姓名} {Wisdom|智慧},{speak|  
说:agent={~}},{think|思考:agent={~}}}
```

当一个语义类在知识词典中被定义为与概念相同的形式时, Step 5.4 给出了与 Step 5.3 相同的计算, 也就是, 检测在两个义原类定义之间的描述节点的相似度, p_4 被给出。

Step 5.5. 相似度的最终计算将基于如下内容。

$$S(c_1, c_2) = (p_1 * \beta_1 + p_2 * \beta_2 + p_3 * \beta_3 + p_4 * \beta_4) * y \quad (3)$$

β_1 、 β_2 、 β_3 、 β_4 是计算不同阶段的权重。它们的关系如下。

$\beta_1+\beta_2+\beta_3+\beta_4=1$ ，并且它们的取值分别为： $\beta_1=0.1$ ， $\beta_2=0.1$ ， $\beta_3=0.7$ ， $\beta_4=0.1$ 。

当两个给出的概念在同一层次上具有相同的事件义原时，并且如表 7.9 所示，在事件义原下存在语义角色的比较对，那么惩罚因子 y 会被应用于公式 (3) 的解决方案中，HowNet 指定 $y=0.35$ 。相反，指定 $y=1.0$ 。语义角色的比较对如表 7.9 所示。

表 7.9 语义角色的比较对

原始语义角色	被影响的语义角色
Agent Experiencer Existent Possessor	Patient
	Target
	Content
	Partner
	PartnerProduct
	PartnerContent
	Possession

下面给出几个语义计算的例子。

以"doctor"和"nurse"之间的相似度计算为例，这两个词的 DEF 如下。

```
" doctor " :
DEF={human| 人 :HostOf={Occupation| 职位 },domain={medical| 医 },{doctor| 医治 :
agent={~}}}}
" nurse " (护士) :
DEF={human| 人 :HostOf={Occupation| 职位 },domain={medical| 医 },{TakeCare| 照料 :
agent={~} }}
```

根据 Step 5.1，结果为： $p_1=0.000\ 000$ 。

根据 Step 5.2 的公式 (1)，结果为： $p_2=a/(d+a)=1.6/(0+1.6)=1.000\ 000$ 。

根据 Step 5.3 的 $p_3=Ns*2/(Nc1+Nc2)$ ，计算的结果如下。

根据表 7.10 中"doctor"和"nurse"之间 DEF 的比较， $Ns=3$ ， $Nc1=5$ ， $Nc2=5$ 。

$$p_3 = Ns*2/(Nc1+Nc2) = 3*2/(5+5) = 6/10 = 0.600000.$$

描述节点之间的比较结果 (二) 如表 7.10 所示。

表 7.10 描述节点之间的比较结果 (二)

"doctor"的描述节点	"nurse"的描述节点
ZeroRole={human 人}	ZeroRole={human 人}
HostOf={Occupation 职位}	HostOf={Occupation 职位}

续表

"doctor"的描述节点	"nurse"的描述节点
domain={medical 医}	domain={medical 医}
ZeroRole={doctor 医治}	ZeroRole={TakeCare 照料:agent={~}}
Agent={~}	Agent={~}

如表 7.10 所示,那么 Step 5.4 如下。"doctor"和"nurse"的范畴义原是相同的:"human|人",其在分类结构中的 DEF 如下。

```
{AnimalHuman|动物:HostOf={Ability|能力}{Name|姓名}{Wisdom|智慧},{speak|说:agent={~}},{think|思考:agent={~}}}
```

{human|人}的描述节点如表 7.11 所示。

表 7.11 {human|人}的描述节点

描述节点 {human 人}
ZeroRole={Animalhuman 动物}
HostOf={Ability 能力}
HostOf={Name 姓名}
HostOf={Wisdom 智慧}
ZeroRole=,{speak 说}
Agent={~}
ZeroRole=,{think 思考}
Agent={~}

如表 7.11 所示,其结果如下。

```
p4 = Ns*2/(Nc1+Nc2) = 8*2/(8+8) = 1.000000..
```

两个给出的词汇在同一层次有同一事件义原,并且在表 7.11 中,在事件义原下不存在语义角色的比较对, $y = 1.0$ 。最终,"doctor"和"nurse"之间的相似度如下。

```
S(doctor,nurse) = (p1*β1 + p2*β2 + p3*β3 + p4*β4)* y
= (0.000000*0.1+1.000000*0.1+0.600000*0.7+1.000000*0.1)* 1.0
= (0.000000+0.100000+0.420000+0.10000)* 1.0
= 0.620000.
```

以"doctor"和"patient"之间的相似度为例,"doctor"和"patient"的 DEF 如下。

```
" doctor ":
DEF={human|人:HostOf={Occupation|职位},domain={medical|医},{doctor|医
治:agent={~}}}}
" patient "(病人):
```

DEF={human|人:domain={medical|医},{SufferFrom|罹患:experiencer={~}},{doctor|医治:paitent={~}}}

根据 Step 5.1 给出的结果计算： $p1 = 0.000\ 000$ ，并且根据 Step 5.2 给出的公式（1）计算如下。

$$p2 = a / (d+a) = 1.6 / (0+1.6) = 1.000000.$$

得到 Step 5.3，根据这个公式计算： $p3=Ns*2/(Nc1+Nc2)$ 。

根据表 7.11，得到： $Ns = 3, Nc1 = 5, Nc2 = 5$ 。

$$p3 = Ns*2 / (Nc1+Nc2) = 3*2 / (5+5) = 6/11 = 0.545455.$$

描述节点之间的比较结果（三）如表 7.12 所示。

表 7.12 描述节点之间的比较结果（三）

"doctor"的描述节点	"patient"的描述节点
ZeroRole={human 人}	ZeroRole={human 人}
HostOf={Occupation 职位}	
domain={medical 医}	domain={medical 医}
ZeroRole={doctor 医治}	ZeroRole={doctor 医治}
Agent={~}	Patient={~}
	ZeroRole={SufferFrom 罹患}
	experiencer={~}

如表 7.12 所示，Step 5.4 如下。"doctor"和"patient"的范畴义原是相同的："human|人"，其在分类中的 DEF 如下。

{AnimalHuman|动物:HostOf={Ability|能力}{Name|姓名}{Wisdom|智慧},{speak|说:agent={~}},{think|思考:agent={~}}}

结果为： $P4 = Ns*2/(Nc1+Nc2) = 8*2/(8+8) = 1.000\ 000$ 。

两个给定的词在同一层次具有相同的事件义原，并且在表 7.13 中，在事件语义下存在一个语义角色的比较对， $y=0.35$ 。

描述节点之间的比较结果（四）如表 7.13 所示。

表 7.13 描述节点之间的比较结果（四）

ZeroRole={doctor 医治}	ZeroRole={doctor 医治}
Agent={~}	Patient={~}

根据 Step 5.5 可得出如下结果。

```
S(doctor,patient) = (p1*β1 + p2*β2 + p3*β3 + p4*β4)* y
= (0.000000*0.1+1.000000*0.1+0.545455*0.7+1.000000*0.1)* 0.35
= (0.000000+0.100000+0.3818185+0.100000)*0.35
= 0.5818185*0.35
= 0.203636.
```

总结一下，在"doctor"和"patient"之间的相似度不同于"doctor"和"nurse"之间的相似度。应该确信，知网 CSM 得到的结果是合理的并且是可以接受的。

- ❑ 在"doctor"和"nurse"之间的相似度为 0.620 000。
- ❑ 在"doctor"和"patient" 之间的相似度为 0.203 636。

7.4 语义网与百科知识库

HowNet 的发布一度成为人工构建汉语知识库的标准。近些年，随着互联网的兴起，人们不再满足于中小规模的人工知识库的构建，而是转向大规模的、自动知识库的构建理论方向。本节以维基百科为例，从理论到实践介绍了语义网理论和目前较大规模的百科知识库。

7.4.1 语义网理论介绍

语义网（Semantic Web）是由万维网联盟的蒂姆·伯纳斯·李（Tim Berners-Lee）在 1998 年提出的一个概念。当时，作为对未来网络的一个设想，它的核心是，通过给万维网上的文档（如 HTML 文档、XML 文档）添加能够被计算机所理解的语义“元数据”（Meta Data），从而使整个互联网成为一个机器可读的、通用的信息交换媒介。也就是说，语义网设计目标是使计算机能够理解词语和概念，而且还能够理解它们之间的逻辑关系，并凭此进行逻辑推理。

语义网概念的提出具有重要的意义。一方面，它为海量的万维网资源设计了机器可读、可处理的美好前景；另一方面，它借助原有的知识库理论（类似 WordNet 和 HowNet），作为实现这个前景的现实方法。应该讲，语义网概念的提出将传统的、狭窄的领域知识库理论技术扩展到了互联网范围。

为此，蒂姆·伯纳斯·李提出了最初的语义网体系结构，图 7.23 给出了语义 Web 的体系结构，各层的功能自下而上逐渐增强。

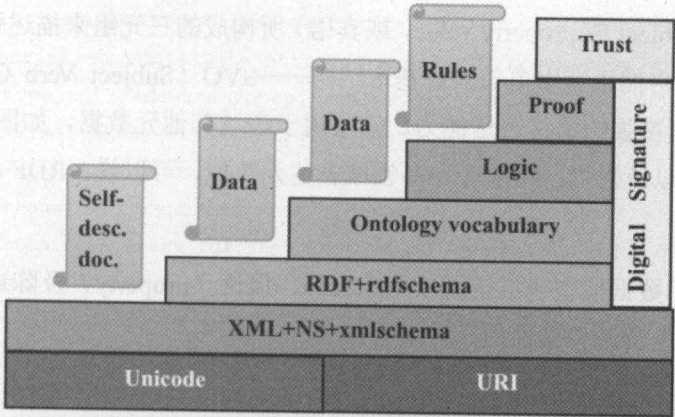


图 7.23 语义网体系结构

如图 7.23 所示，第 1 层。基础层，主要包含 Unicode 和 URI（Uniform Resource Identifier）。它们是语义的字符编码和所属的网络标识，是构成语义网所必需的前提。Unicode 代表了一种字符编码体系；URI 类似各种资源的定位 ID，是构成语义节点最基本的要素。

第 2 层：句法层。从蒂姆·伯纳斯·李的理论来看，是一系列由 XML 定义的相关规范，如 XML 文件、命名空间等。但从实践来看，是一系列使用自然语言方式编写的、适合人类阅读的网页文本。第 2 层的主要目标是这类文本解析为最基本、最完整的语义形式，为下一层编码为机器可读的 RDF 做准备。

为了更好地理解，下面以一个自然语言文本转换为 RDF 的实例来说明。假设有几个句子，它的原始形式可能是一个复句，也可能是一个段落，等等。把这些句子分拆，变为最简单的 SVO 形式，如表 7.14 所示。

表 7.14 RDF最简SVO列表

<subject>（主语）	<predicate>（谓词）	<object>（宾语）
<Bob>	<is a>	<person>
<Bob>	<is a friend of>	<Alice>
<Bob>	<is born on>	<the 4th of July 1990>
<Bob>	<is interested in>	<the Mona Lisa>
<the Mona Lisa>	<was created by>	<Leonardo da Vinci>

第 3 层：RDF 层。资源描述框架（Resource Description Framework，RDF）是一种用于描述万维网资源信息的通用框架，如网页的内容、作者以及被创建和修改的日期等。RDF 本质上是一种数据模型，由主体（subject，或主语）、谓词或属性（predicate 或 property）、

客体或属性值 (object 或 property value, 或宾语) 所构成的三元组来描述资源的元数据。RDF 的定义很像自然语言中单句的最基本结构——SVO (Subject Verb Object)。所以, RDF 天生就具有很强的语义表达能力。它可用于表达其他元数据, 如图书的书目信息、自然语言句子, 以及生物、化学等许多领域表达元数据。可以说, RDF 已经成为知识表达的通用形式。

RDF 的基本数据模型包括资源 (resource)、属性 (property) 及陈述 (statements)。确切地说, 陈述是资源和属性的一种组合。

- ❑ 资源。一切能够使用 RDF 表示的对象都称为资源, 包括网络上的所有信息、虚拟概念和现实事物等。资源用唯一的 URI 来表示, 不同的资源拥有不同的 URI, 通常使用的 URL 只是它的一个子集。
- ❑ 属性。用来描述资源的特征或资源间的关系。每一个属性都有其意义, 用于定义资源的属性值、描述属性所属的资源形态, 以及与其他属性或资源的关系。
- ❑ 陈述。一条陈述包含三个部分, 通常称为 RDF 三元组<主体, 属性, 客体>。其中, 主体是被描述的资源, 用 URI 表示。客体表示主体在该属性上的取值, 可以是另外一个资源 (由 URI 表示) 或者一个文本。

接上例, 将上述表格转换为语义图 (见图 7.24) 的形式展示出来。

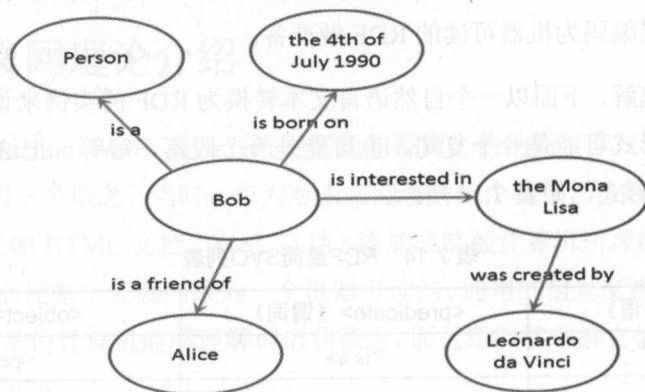


图 7.24 语义图

最后, 生成机器可读的 XML 格式的 RDF, 代码如下。

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

```

```

    xmlns:schema="http://schema.org/">
    <rdf:Description rdf:about="http://example.org/bob#me">
      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
      <schema:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
1990-07-04</schema:birthDate>
      <foaf:knows rdf:resource="http://example.org/alice#me"/>
      <foaf:topic_interest rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.wikidata.org/entity/Q12418">
      <dcterms:title>Mona Lisa</dcterms:title>
      <dcterms:creator rdf:resource="http://dbpedia.org/resource/Leonardo_da_Vinci"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://data.europeana.eu/item/04802/ 243FA8618938
F4117025F17A8B813C5F9AA4D619">
      <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    </rdf:Description>
  </rdf:RDF>

```

因为篇幅有限,这里对生成的 RDF 不做进一步解释,希望进一步了解 RDF 规范的朋友可以从 <http://www.threedweb.cn/thread-1436-1-1.html> 下载详细的规范内容。

第4层:本体层。本体 (Ontology) 一词源于哲学领域,指事物的本身,引申为根本。在计算机科学领域,是指一种模型,用于描述由一组对象的类型 (概念或者类)、属性及关系类型所构成的世界。有关本体的详细内容与 HowNet 中概念构建的差异不大,都有属性、属性值、Function (事件) 等要素。目前,本体的构建仍旧依赖于手工处理,大规模的本体自动构建距离还比较遥远,因此不是本文的重点。有关本体的更多细节,建议参考 W3C 组织提供的相关规范和教程文档,读者可从如下网址下载:
<http://nkos.lib.szu.edu.cn/OWL2/OWL2PrimerSimplifiedChinese.htm>

第5~7层分别是逻辑层 (Logic)、验证层 (Proof) 和信任层 (Trust)。逻辑层在前面各层的基础上进行逻辑推理操作。验证层根据逻辑陈述进行验证,以得出结论。信任层是语义网安全的组成部分,与加密不同的是,该层主要负责发布语义网所能支持的信任评估。目前第6层和第7层正处于设想阶段。

由于 RDF 三元组是语义网数据表示的基础。要实现从万维网到语义网的转变,构建海量的 RDF 数据集是一项基础性工作。虽然 W3C 定义了 RDFs、OWL 等语义网规范,但互联网中的绝大多数内容都是自然语言形式的文本,尚未转换为符合语义网规范的形式。因此,如何自动化地从现有的 Web 内容中抽取出符合语义网规范的语义内容是语义网走向实用化面临的难题之一。

不同的公司为此开发了一系列的技术,涉及信息抽取、分类、表达、存储、查询等。其中,这些技术的核心仍旧是自然语言处理的相关技术,即如何准确地解析网页中的句子,并把句子映射为 RDFs 的格式。乍一看,只要有准确的分词、词性标注、句法解析系统即可,似乎这个过程并不复杂。换句话说,只要句法解析系统的精度达到商业化的要求,这个目标即可实现。但在实践中,语义表达的方式和内容却复杂得多。完全自动化地解析映射,都不会达到很好的效果。语义网正如它的名字,如果不对语义本身有透彻的认识,想要实现 RDFs 或 OWL 的自动生成,恐怕只是一个美好的前景。

7.4.2 维基百科知识库

不过,这些公司的努力并没有白费。很多新的网络服务形式和信息挖掘模式被开发出来,其中最引人注目的是对百科知识库的信息挖掘。在所有大型的百科知识库中维基百科(Wikipedia)是历史最悠久、影响范围最大的一个。

维基百科是一个内容公开、自由编辑且多语言的网络百科全书协作项目,通过 Wiki 技术使得包括用户在内的所有人都可以简单地使用网页浏览器修改其中的内容。维基百科一词取自网站核心技术“Wiki”(一种可供多人协同写作的网络技术),以及具有百科全书之意的“Encyclopedia”,共同创造出新混成词“Wikipedia”,当前维基百科由维基媒体基金会负责营运。维基百科是以众包形式,由网络志愿者共同合作编写而成的,任何使用互联网进入维基百科者都可以编写和修改里面的文章。与传统的百科全书相比,在互联网上运作的维基百科其文字和绝大部分图片使用 GNU 自由文件许可协议和知识共享署名的方式共享 3.0 协议来提供对每个人来说自由且免费的信息,任何人都可以成为条目的作者,以及在遵守协议并标示来源后直接复制、使用及发布这些内容。

以中文维基百科为例,迄今为止,中文维基百科已有 889 661 个条目,但这个数量还不包括对话页、没有内部链接的条目、重定向页及其他名字空间的页面,如果加上这些页面,截至 2016 年 7 月,总共有超过 472 万页,并有超过 4 210 万次的编辑。参与人数方面,共有逾 2 274 930 名注册用户曾进行编辑,其中编辑超过 1 000 次的维基人共有 1 154 人以上,而他们的贡献约占中文维基百科编辑总次数的 73.6%。

值得庆幸的是,由于维基百科来源于一个众包项目,任何用户都可以下载其全部资源。数据资源的下载说明可从 URL: <https://zh.wikipedia.org/zh-cn/Wikipedia> 的下载链接中找到,仅中文版就提供了如下 8 个不同的版本。

❑ 中文版的下载地址: <http://download.wikipedia.com/zhwiki/>。

- ❑ 文言文版的下载地址: http://download.wikipedia.com/zh_classicalwiki/。
- ❑ 粤语版的下载地址: http://download.wikipedia.com/zh_yuewiki/。
- ❑ 吴语版的下载地址: <http://download.wikipedia.com/wuuwiki/>。
- ❑ 赣语版的下载地址: <http://download.wikipedia.com/ganwiki/>。
- ❑ 客家话版的下载地址: <http://download.wikipedia.com/hakwiki/>。
- ❑ 闽南语版的下载地址: http://download.wikipedia.com/zh_min_nanwiki/。
- ❑ 闽东语版的下载地址: <http://download.wikipedia.com/cdowiki/>。

最新的简体中文版维基百科数据资源可以从 <https://dumps.wikimedia.org/zhwiki/latest/> 下载。所有数据文件都使用 bzip 2、gz 或 7-zip 进行了压缩。解压后的文件主要有两种格式: xml 和 sql。xml 文件中的内容是以 RDFs 格式保存的三元组, sql 格式中是对应的网页数据。

国内用户无法正常登录维基百科。因此, 维基百科提供了一个本地浏览版的软件——WikiTaxi (<http://www.wikitaxi.org/delphi/products/wikitaxi/index>), 有兴趣的读者可以使用这个软件从本地浏览中文维基百科的文件。

7.4.3 DBpedia 抽取原理

DBpedia 是语义网在 RDF 和 OWL 层的应用范例, 它能从维基百科的词条里抽取出结构化的数据 (RDF 和 OWL), 强化了维基百科在语义方面的搜寻功能, 并将用户的数据集链接到维基百科。通过专门的语义分析和提取技术, 使维基百科的庞杂知识有了许多创新而有趣的应用, 如手机版本、地图整合、多面向搜寻、关系查询、文件分类与标注等。DBpedia 有如下几个重要的特点。

- ❑ DBpedia 采用 RDF 语法表示和组织知识, 并支持基于 SPARQL 语法的知识查询。目前, DBpedia 知识库已包含超过 10.3 亿 RDF 三元组。
- ❑ 采用 OWL 语言创建了 DBpedia 本体, 更好地支持基于语义 Web 的知识组织活动。在维基百科中, 元数据以信息框 (InfoBox) 的形式记录和保存, 不同的信息框应用频度可能不同。DBpedia 从这些 InfoBox 中找出一些最常用的数据项, 分析相互关系, 用手工方式创建原始数据集的本体库。
- ❑ 为了对数据集进行语义分类, 提高分类效果, DBpedia 支持 4 种知识分类方法: 维基百科分类方法, 可支持多达 41.5 万个类; YAGO 分类方法, 可支持多达 28.6 万个类; UMBEL (Upper Mapping and Binding Exchange Layer) 分类方法, 可支持多达 2 万个类型; DBpedia 本体, 支持 170 个大类和 720 个属性。

2014 版的 DBpedia 知识库已经拥有超过 458 万个物件，包括 1 445 000 人、735 000 个地点、123 000 张唱片、8 700 部电影、19 000 种电脑游戏、241 000 个组织、251 000 种物种和 6 000 个疾病。其资料不仅被 BBC、路透社、纽约时报所采用，也是 Google、Yahoo 等搜索引擎检索的对象。最新版的 DBpedia 全部数据可以从 <http://downloads.dbpedia.org/2015-10/core-i18n/zh/> 下载。为了便于用户的使用，DBpedia 还在 GitHub 上发布了一系列的项目，下载网址：<https://github.com/dbpedia/>。

在所有的应用中，最引人注目的是 DBpedia 的抽取架构，如图 7.25 所示。

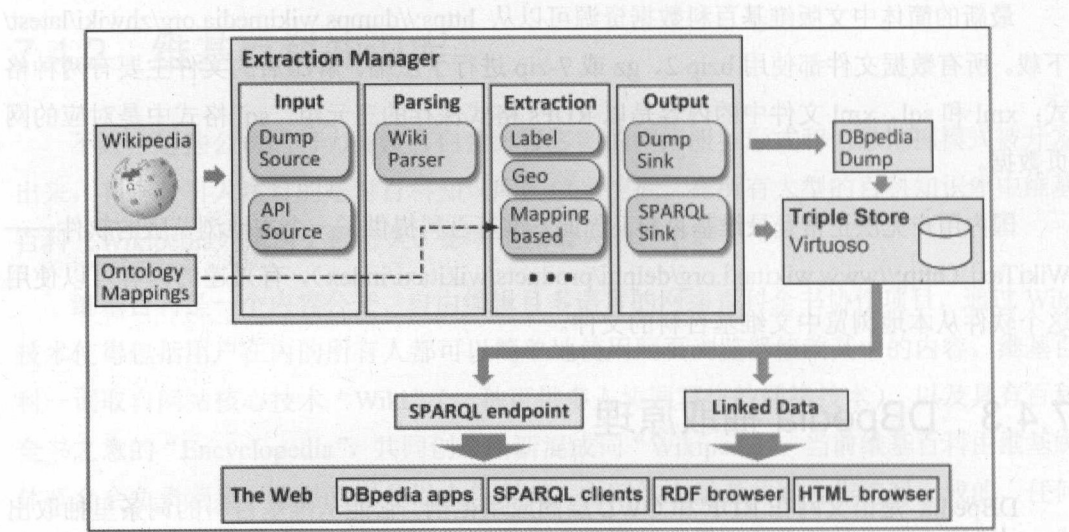


图 7.25 DBpedia 抽取架构

图 7.25 中部分内容说明如下。

- ❑ Input（输入）：读取维基百科页面作为输入。
- ❑ Parsing（解析）：通过一个专门的维基解析器（Wiki Parser）来解析每个维基百科的页面。维基解析器将维基百科的页面转换为一个抽象的句法树（Abstract Syntax Tree）。
- ❑ Extraction（抽取）：每个维基百科页面的抽象句法树都被转移到一个抽取器中。DBpedia 为不同类型的文档提供了不同的抽取器，如抽取标签、摘要、地理坐标等。
- ❑ Output（输出）：将收集到的 RDF 语句写入一个容器中，然后根据需求转换为不同的 RDF 格式，如 N-Triples 等。

需要进一步说明的是有关 DBpedia 抽取器的一些设计细节，DBpedia 提取的方式主要可以分成如下四大类。

- ❑ 原 Infobox 抽取 (Raw Infobox Extraction): 直接将维基百科条目中的 Infobox 转换为 RDF (见图 7.26)。

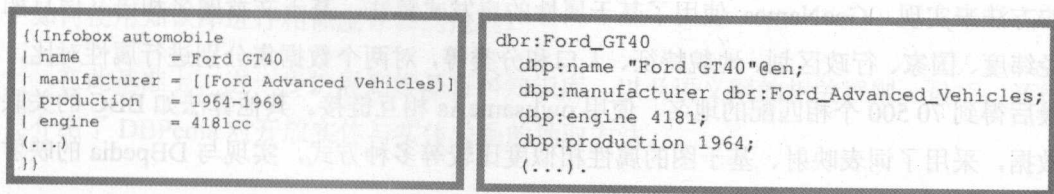


图 7.26 Infobox 抽取 RDF

- ❑ 基于映射的 Infobox 抽取 (Mapping-Based Infobox Extraction): 手动编写 Infobox 到维基百科在 DBpedia 本体的映射关系。需要注意的是, 所有值都需要规范化为基本单位。

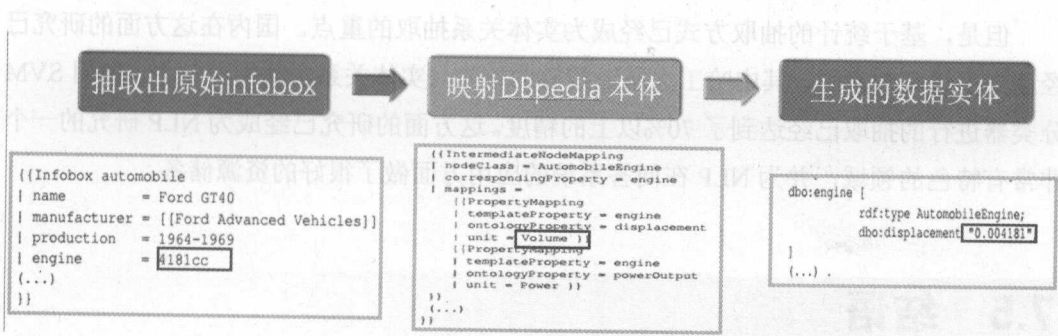


图 7.27 Infobox 映射本体

- ❑ 特征抽取 (Feature Extraction): 采用多种抽取器, 其中的每一种都只从文本中提取一个单一的特征, 如地理坐标或一些标签。

如果有明显的、特定的命名模式, 则可以使用特点模式抽取器。在各种领域, 有各自普遍使用的命名模式, 如出版界有 ISBN 号、金融界有 ISIN 号, 如果这些标识被用作 URI 的一部分来标识特定的资源, 那么就可以使用简单的模式匹配算法生成 RDF 不同资源之间的 RDF 链接。例如, RDF 的 BookMashup 标签采用 ISBN 号作为 URI 的一部分, 《哈利波特与混血王子》一书的 URI 为 <http://www4.wiwiwss.fu-berlin.de/bookmashup/books/0747581088>。DBpedia 遍历所有图书的 ISBN 号, 生成一个与 Book Mashup 相应的 URI, 并创建 DBpedia 图书与生成的 URI 之间的 owl:sameAs 链接。应用此算法, DBpedia 产生了 9 000 个内部 RDF 链接。

- ❑ 统计抽取 (Statistical Extraction): 使用了一些有关属性抽取器从页面抽取汇总数据, 以便提供基于网页链接或字数统计的数据。

在没有通用标识符的情况下,将事物的属性特征作为生成自动关联的依据。例如,DBpedia 与 GeoNames 都包含地理名称,要映射两者的重合部分,可通过识别相关属性的方法来实现。GeoNames 使用了基于属性的启发式算法,基于文章题名和语义信息如经纬度、国家、行政区划、地貌特征、人口和分类等,对两个数据集分别进行属性对比,最后得到 70 500 个相匹配的地名,使用 owl:sameAs 相互链接。其他算法如 BBC 的关联数据,采用了词表映射、基于图的属性相似度比较等多种方式,实现与 DBpedia 的映射链接。

由此可见,虽然 DBpedia 的规模已经非常庞大,但抽取内容依然以结构化数据(InfoBox)为主,统计抽取所占的比重并不大。这说明自动化语义解析的工作仍旧处于一个实验的阶段。

但是,基于统计的抽取方式已经成为实体关系抽取的重点。国内在这方面的研究已经达到了较高的水平,其中哈工大车万翔等的论文《实体关系自动抽取》中,使用 SVM 分类器进行的抽取已经达到了 70% 以上的精度。这方面的研究已经成为 NLP 研究的一个非常有特色的领域,并为 NLP 在问答系统的应用方面做了很好的资源储备。

7.5 结语

本章从历史、设计、分析、处理、应用等方面,全面介绍了 NLP 领域常用的一些语料库,内容详实、资源丰富。本章涉及的语料库包括 NLP 基础任务所使用的语料资源:分词和词性标注语料库、大规模汉语树库,以及这些语料库的制作方式;还包括语义知识库,如 HowNet 知识库和维基百科知识库。

在基础任务语料库建设方面,我们以国家语委语料库为例,详细介绍了语料库构建的时间、规模、语料的选择、软件功能等方方面面的知识。只要用户有一定的程序设计基础,通过学习本章内容后完全可以设计出自己的语料库。

在已完成的基础语料库方面,本章给出了如下几个著名的基础语料库。

- ❑ 生语料库:搜狗互联网语料库(超大规模)。
- ❑ 分词和词性标注语料库:1998 年人民日报语料库、MSR 微软中国研究院语料库。
- ❑ 汉语树库:宾州大学中文树库 CPB 8.0。

在知识库方面,我们详细分析了 HowNet 语义知识库的方方面面,内容包括:义原的发掘、语义角色列表、实体与事件分类、属性与属性值的定义。并通过一个实例介绍了如何使用知识库进行相似度计算的过程。

在此基准上,进一步扩展到维基百科知识库,以及语义网的相关规则。最后,还简要介绍了 DBPedia 对开放实体与实体关系的抽取方法。

第 8 章

语义与认知

什么是意义，什么是意义的意义？传统上除语言学外，意义常在哲学领域中被讨论。所以，“意义”绝不是一个简单的问题。最显著的问题就是对语言的任何一种解释都必须也要用语言（常换作其他的词汇或其他的语言）。这就无可避免地进入了循环解释的悖论中。比如，我们想要解释“上”，新华词典的解释如下。

“‘上’ shàng 位置在高处的，与‘下’相对：楼上。上边。”

按照这个解释，如果我们想要理解“上”，就必须要知道词汇“位置”、“高处”的含义，更要理解“下”的含义。可是，一个人如果连“上”都不知道，又有多大可能知道其他词的含义呢？这种解释本身没有任何意义。

长久以来，只要人们想要分析某个事物，从科学的视角上，都习惯使用还原论的思想。例如，现代庞大的数学学科体系就是从几条不可证明的公理发展而来的，前面介绍过的短语结构语法就是这样发展起来的。以物理学为首的各种理工类学科都是比较典型的例子。我们希望语言学，特别是对语义的研究，也符合这一规律。通过论证语义的最基本形式来推导出丰富多彩的词汇的意义。

因此，用某些最基本的词汇作为语义的义素来解释其他的词汇，这种思想几乎统治了一个多世纪的语言学界，并且人们为此付出了很大的代价。现在我们知道，语义本身并不贴合还原论的思想。当今的语义学的分析方法已经作为一种横断的方法论，内在地融入到诸多学科发展的发展趋势之中，其系统的、综合方法论的地位毋庸置疑。下面通过本章的简要介绍逐渐揭开语义的奥秘。

8.1 回顾现代语义学

所谓现代语义学，其实是指认知语言学产生之前的，上世纪以来的主要语义学方面的理论。虽然我们称之为“现代”，但实际上是将“当前”定位为“后现代”而言的。

8.1.1 语义三角论

19 世纪后半叶及 20 世纪前几十年中，语言学和哲学十分关心语言与感知、物理世界的关系，关心词语的来源，以及语言结构中的心理过程、概念方式和过程等。由英国学者奥格登 (Ogden) 和理查兹 (Richards) 在 1923 年出版的语义学的重要著作——《意义的意义》(The Meaning of Meaning) 中首次提出了“语义三角论”的概念。它代表了传统语义学的典型观点。语义三角论是一种全新的意义模式，也称为意义三角论 (见图 8.1)。

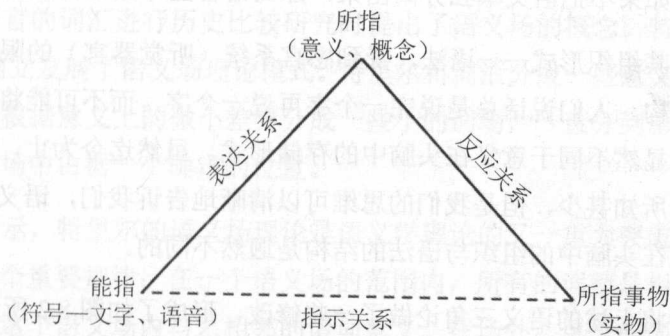


图 8.1 意义三角论

该理论是指符号、意义和客观事物之间处于一种相互制约、相互作用的关系之中。它强调语言符号是对事物的指代，指称过程就是符号、意义和事物发生关系的过程。

语义三角论包含如下几点含义。

(1) 概念/思想 (concept/thought) 和客观事物 (referent/thing) 之间是直接的联系。概念是在客观事物的基础上概括而成的，是客观事物在头脑中的反映。二者用直线连接，表示 a concept refers to a thing，即概念反映客观事物。

(2) 概念与符号/词 (symbol/word) 之间也有直接联系。概念是通过符号表达出来的，二者用实线连结，表示 a word symbolize a concept，即词表示概念。

(3) 符号或词与指称物/事物 (referent/thing) 之间没有直接的、必然的联系，二者

之间具有任意性,是约定俗成的。虚线表示 a word stands for a referent,即词代表指称物。符号与指代物之间没有内在的必然联系,真正的联系存在于人的头脑之中。

——来自百度百科《语义三角》词条

语义三角论的提出是人类语言学史上的一个重要的突破,它使语义研究从语言系统中独立地分离出来,是此后语法学(符号学和结构学)、语义学和语境学(也称为语用学,这里提到语境是为了后面的理论做准备)三足鼎立局面的一个重要开端。

语义三角论揭示了一个重要的原理:意义来源于人们对外界对象的一种反射,这种反射可以是感性的,也可以是理性的,但都是人性的;可以是片面的,也可以是整体的,但都是一种印象;而语言符号是意义的载体,是对意义的一种编码,具体符号的表示方式因人们所说的语言不同而具有差异性,但语义本身对人类都是共通的。因此,意义是语言的灵魂,失去了意义,语言就失去了表达的内容。这就是语义先决性论断的基础。

其实在现实中,语义的共通性与符号差异性是不言自明的。中国人说汉语,英国、美国人说英语,如果不把语义单独分离出来,那么语言翻译又如何成为可能!

语言符号及其组织形式——语法,受到感官系统(听觉器官)的限制,是按时间序列排列的顺序结构。人们说话总是说完一个字再说一个字,而不可能将所有的字都同时说出来,但是这显然不同于意义在头脑中的存储形式。虽然迄今为止,我们对大脑如何存储知识的细节所知甚少,但是我们的思维可以清晰地告诉我们,语义的结构并不受时空的限制,知识在头脑中的组织与语法的结构是迥然不同的。

为此,笔者将上述的语义三角论做了一些修改,形成了如图 8.2 所示的模型。

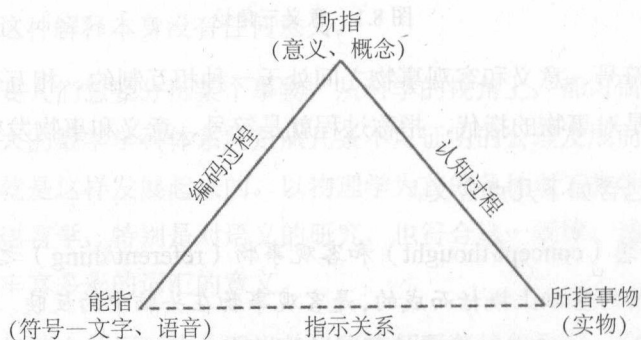


图 8.2 认知科学下的意义三角论

图 8.2 的另一个变化是,将“反应关系”修改为“认知过程”。所谓认知过程是指人类认识客观事物的过程,即人对信息加工处理的过程,这一过程表现为认识的由表及里、由现象到本质的过程。这个过程因为由人的感觉、知觉、记忆、思维和想象等认知要素

所组成，所以它是一种客观事物特征与内在联系在人脑中反映的心理活动。当然，也伴随着心理活动中的心理特征。

意义的获得器官主要来自视觉，之后是听觉、触觉、味觉等，然后通过神经系统传递到大脑进行记忆、编/解码、联想、抽象、推理、判断等分析或综合的思考过程形成某种知识结构存储起来。语义的本质就是人们对外界对象的一种认知。也就是说，我们所表达的一切都是我们对外界对象认知过程的一个片段。可能是表象，也可能是本质；可能是具体，也可能是抽象，所以丰富多彩、千差万别。经过了大脑加工的意义，并不是客观的对象本身。因此，只有充分地了解语言的认知过程，才能准确地理解语言的意义。而汉语又是一种强表义的语言，没有对语义及其结构的深刻了解，想要实现汉语的自动处理几乎是不可能的。

8.1.2 语义场论

几乎同时，1924 年，德国语言学家伊普森（G .IPsen）也从语义的视角来研究语言，在其对古印欧语言的词汇进行历史比较研究时提出了语义场的概念。特里尔（Trier）在 1934 年进一步确立发展了语义场理论模式。特里尔将词汇分成一些意义相近的大的词场，这些大的词场再根据意义上的微小差异分成一些小的词场，一直分到单个的词。这样，每个词就在概念场中占据一个确定的位置。

如图 8.3 所示，特里尔的语义场理论是语义学理论的又一重大突破，他的理论发现了词汇语义的一个重要规律：在一个语义场的范围内，所有的词都是相互联系的，每个词的意义取决于这个语义场内与之相邻的诸词意义。在场外，单个的词没有任何内容，即或有，其意义也是极不确定的。只有在语义场中，词才获得单义性，在言语中才能被人理解。例如，特里尔指出，德语 *weise*（英明的），由于存在如下一些同属一个语义场的词：*kiug*（聪明的）、*gerissen*（敏捷的）、*schlau*（灵巧的）、*gwitzigt*（伶俐的），其含义也会有所不同。

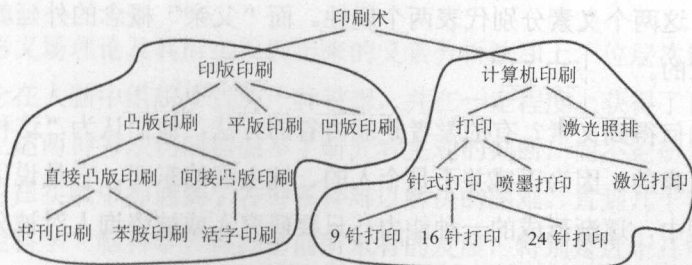


图 8.3 “印刷术”下的两个子场示例

特里尔的理论开创了语义学史上的新阶段。其后很长一段时期,人们对多种语义场(如对颜色、亲属、官衔等语义场)做了详尽的研究和描写。在描写过程中,人们也发现对所描写的语义场做出准确的界定十分必要。但界定的标准涉及两个方面:一方面要确定所要研究的语义场自身的规模;另一方面也要确定一个语义场在其父类层次结构中的位置。

1. 语义场的自身规模

有关单个语义场的规模研究,最终导致词汇与概念的分离。语义场的核心构成单位应是概念,词语是围绕着概念进行聚合的语言符号。以“亲属语义场”为例,在进行语义场分析时,与“父亲”同义的亲属词就有“爸爸”、“家翁”、“爹”、“老子”、“阿爸”、“家父”等,与“岳父”同义的亲属词有“岳丈”、“岳翁”、“丈人”、“泰山”、“冰翁”等,如果将“父亲”、“岳父”作为同义集合的代表,经过筛选,可以得到的语义场大致由 53 个同义词构成。

这样就出现了一个问题,如何表示“父亲”这个概念呢?因为词汇和概念产生了分离,用单个词汇表示概念显然是有偏差的。这种现象与逻辑学中关于概念的定义相一致。概念分为内涵和外延两部分。词汇的内涵是它的义项即概念,是事物的特有属性的反映,如“父亲”的内涵,其词典的义项为“有子女的男子是子女的父亲”。词汇的外延是词项所指的事物共同构成的类,也称为范畴,即所在的语义场。如“父亲”的外延就是上段提到的所有同义词汇。

为了能更精确地界定一个语义场,人们对词汇的语义关系进行进一步的分析。这种分析甚至深入到词义内部结构。这直接导致了义素分析方法的提出,也反过来促使了人们在一个更加具体、清晰的层面对语义场进行定义:语义场实质上是由具有某些共同义素的词群聚合而成的场。

“义素是构成词义的最小意义单位,也就是词义的区别特征”(黄伯荣、廖序东 1983)。如上文中“父亲”一词,根据现代汉语词典的义项至少可将其分解为“有子女的”、“男人”两个义素。这两个义素分别代表两个属性。而“父亲”概念的外延就是通过这两个义素的交集确定的。

问题在于如何得到义素?有的学者是靠内省的方法,他们认为“这种靠语感来分析义素的方法是可靠的,因为语感并不是个人的、纯主观的东西,它是说话人在长期使用某一语言的过程中,逐渐形成的一种能力;只要研究人或被咨询人对被分析的对象十分熟悉,就可以试着采用这种比较方法(贾彦德 1999: 64)”。

更方便的做法是依靠词典的定义。一般的词典在揭示词义时往往把它的属性排列出来,首先通过类属关系同别的非本类事物相区别,再根据某些特征同本类内部其他事物相区别。但在实践中,无论这些方法是否可靠、客观,都较难施行,且成本很高。对于海量的词汇而言,靠人工自省确定义素是一项烦琐而艰苦的工作。下文介绍的 HowNet 知识库也采取了这两类方法来定义语义知识库,我们将以此例来详细说明。

2. 语义场的层次结构

人们进一步对语义场的层次性进行考察,发现某些词可以在一个共同概念或语义成分的支配下形成语义场。表达共同概念的词称为上义词(Superordinate,上位词),语义场由上义词和若干个受上义词支配的下义词(Hyponym,下位词)组成。如“动物”一词,作为上义词,与其支配的“鸟”、“鱼”、“昆虫”等一系列与其具有共同语义特征的下义词一起组成“动物场”。再如“植物”一词,与受其支配的“花”、“树”、“农作物”等词构成“植物场”。作为上义词的“动物”和“植物”又可受“生物”的支配,转而变成两个并列下义词,一起构成更高一层的“生物场”。上义词和下义词具有明显的相对性。如上述“鸟”、“鱼”和“昆虫”三个词。一方面,相对于“动物”而言是下义词,另一方面相对于“麻雀”和“老鹰”、“鲤鱼”和“鲨鱼”、“蝴蝶”和“蝉”等词而言又都可以成为上义词,构成各自独立的语义场。属于同一个语义场的词在语义上相互依存、相互制约。也就是说要确定某个词的意义,必须首先比较该词与同一语义场中其他词在语义上的联系,以及该词在语义场中所在的位置。

这种研究方法与同时期的分类学不谋而合。分类学(Taxonomy)是区分事物类别的学科。最初的现代分类学是指生物分类学,即研究活着的和已灭绝的动植物分类的科学,研究动物、植物的鉴定、命名和描述,把物种科学地划分到一种等级系统以此反映对其系统发育的了解情况。后来人们将这类方法推广到语言学,试图将表达世界万事万物的概念按照上下位关系进行分类,并建立一个庞大的、统一的分类体系。上下位关系和这个庞大的分类系统都构成了语义知识库的基本要件。下文的 HowNet 语义知识库会对此进行详细说明。

在当时,语义场理论及其后续发展出来的义素分析法和上下位层次结构,都是对语义,特别是概念在人脑中组织形式的一种设想,并在一定程度上获得了良好的效果。但是现在看来,上述两种方法仍旧摆脱不了研究者主观的判断,而不论研究者是群体还是个体,这些理论在实践中都遇到了各种各样难以解决的困难。直到几十年后,生物学和医学领域的神经科学(脑科学)得到了前所未有的发展,特别是近十几年,在认知科学上的新发展,才使该理论逐渐又有了新的突破。

8.1.3 基于逻辑的语义学

在对词的研究告一段落之后,人们的眼光逐渐转向对句子的语义探索。这部分的工作起始于20世纪60年代前后的蒙太古(R. Montague)语义学,也称为蒙太古语法。它是一种应用现代逻辑方法,形式化地处理自然语言语义的理论。其曾经吸引了国际学术界众多学者、名流,从现代逻辑学、现代语言学、当代西方语言哲学、计算机科学诸方面,都对它进行了拓展性的深入研究,并取得了一定的学术成果。曾经被誉为“可以与爱因斯坦广义相对论媲美的”,“与乔姆斯基转换生成语法并列的”,“最为引人注目的”理论。并一度被看作在人工语言和自然语言之间架起的一座“桥梁”,是解决人类最为难以驾驭的、深不可测的语义问题的“希望”。

当然,其发展并非一帆风顺,到了20世纪70年代,蒙太古死后,经过语言学家和哲学家的共同努力,将其发展成为一个独立的学科,并且摒弃了蒙太古对生成语言学的句法学的忽视,强调语义解释和句法结构的统一,从而最终成为生成语言学的语义学分支。这就是著名的形式语义学。

形式语义学认为句子的含义(Sentence Meaning)是词义(Lexical Meaning)根据一定方式组合的产物。既然可以孤立地考察词义,如编纂词典,那么也可以机械地研究组词成句后的句义,而不必考虑句子所处的语境因素。

句子的语义从信息内容上可以分成三种类型:真值条件义、表情感叹义和言语行为义。所谓真值条件义是指某类句子的主要用途在于描述主体(说话者)的外部世界。一个陈述性的句子或者真实地反映了外部世界的某种现象,或者对该类现象做出了不正确的、虚假的描述。用逻辑术语来说,就是是非判断(真假判断)。形式语义学通过研究句子在什么场合下为真、在什么场合下为假来判定这类句子的真值条件。并断言,凡是陈述性的语句都具有真值条件。如果把真值条件当作句子的逻辑语义,那么句子即可用形式逻辑的方法进行演算。

句子的另一个用途是表达语言使用者的情感,助其抒发胸臆。这种句子通常不是人们对外部世界的描述,而是其表达的一种价值观念或主观愿望。所以,这些语句无所谓真,也无所谓假,没有真值条件。

句子的另一个用途是运用言语表示行为、处理事情。它们的范畴属于言语行为,这类句子在场合、方式,语法上都有显著的特征,如汉语中的祈使句就属于这类语言,也称为言导行为。这类句子也无所谓真、假。应该讲是成功与失败。因为言导行为成功与否,取决于诸多的语境因素,包括自然条件和社会条件。

逻辑语义学将重点放到了具有真值判断的陈述性语句上,并发展出了一套完整的以命题逻辑和谓词逻辑为中心的推理系统。这类系统后来被证明在人造语言,特别是计算机程序语言方面具有很大的用途,是程序设计理论的重要组成部分之一。学科的任务也从自然语言的语义解析转到程序语言的编译上,从形式逻辑发展到了数理逻辑。并以数学为工具,利用符号和公式,精确地定义和解释计算机程序设计语言的语义学科。最终逐渐淡出自然语言处理的视野。

8.2 认知语言学概述

认知科学是本世纪世界科学标志性的新兴研究门类,它的兴起是以脑科学(认知神经科学)在近些年获得的巨大的进步为基础的。认知神经科学的最终目的是阐明人类大脑的结构与功能,以及人类行为与心理活动的物质基础和内在机制。在其几个重要的基本目标中,至少有如下三个目标是与智能计算(计算语言学)相关的。

- (1) 揭示神经元间各种不同的连接形式,为阐明行为的脑的机制奠定基础。
- (2) 在形态学和化学上鉴别神经元间的差异,了解神经元如何产生、传导信号,以及这些信号如何改变靶细胞的活动。

- (3) 认识实现脑的各种功能(包括高级功能)的神经回路基础。

在半个多世纪以来,认知神经学获得的各项突破中,与智能计算相关的内容包括:成功揭示出视觉的脑机制原理(第9章会详细探讨),以及人工神经网络对生物神经网络的成功模拟等。

认知科学的这些研究成果,在诸多领域颠覆了人们的传统观念,这其中就包括语言学。认知语言学因此应运而生,它于20世纪80年代后期至90年代开始成型,涉及人工智能、语言学、心理学、系统论等多种学科。标志性的事件是1989年召开了国际认知语言学会议,同年,杂志《认知语言学》出版发行。认知语言学的创立者普遍被认为是乔治·雷可夫(George Lakoff)、马克·约翰逊(Mark Johnson)及朗奴·兰盖克。其中,雷可夫及约翰逊专门研究语言中的隐喻及其与人类认知的关系;而兰盖克的专长在于认知语法。认知语言学针对生成语言学的天赋观提出:语言的创建、学习及运用,基本上都必须能够通过人类的认知而加以解释,因为认知能力是人类知识的根本。

严格意义上说, 认知语言学不是语言学的一门分支学科, 而是认知神经学和语言学的交叉学科。认知语言学一经产生就对传统语言学产生了重大的影响。

首先, 认知语言学与传统语言学对语言的基本认识不同: 认知语言学认为词汇或语言的意义就是在听话人的大脑中激活的一些经验模式, 一次或若干次的神经脉冲。意义的描写仅涉及词汇与大脑(神经网络)的关系, 而不是词与世界之间的直接关系。这一结论进一步证实和发展了语义三角论, 而且更阐明了三角论的生物学和心理学机理。而语言的基本功能, 或者更明确地说, 语义的本质属性是“象似性”。语法的不同结构类型受到不同类型的“象似性”的影响。

其次, 语言所表达的内容是语义, 从计算语言学的角度而言也就是知识, 但是, 词汇符号不能脱离语境而独立表义。换句话说, 词汇只是人们用来表义的工具, 词汇的意义脱离了语境, 可能是不固定的, 甚至可以毫无意义。只有在特定的语境中, 词汇与语义才可以配对, 此时词汇的语义才是唯一的。认知语言学强调了大规模知识库和语料库对语言认知的作用, Google 的 Word2Vec 算法遵循这一原则, 从大规模的语言知识库中成功地学习到词汇的语义, 就是这一观点的一个强有力的证明。意义的辨析从此脱离手工的定义, 走向了机器学习的范畴。这不仅是认知语言学, 也是计算语言学的一大突破。

由于上述两点, 人们揭示出语法多样性的本质, 语法结构的多样性同时受到两个因素的影响, 即人们表义的多样性和语境的不同类型的双重影响。

第三, 认知语言学的研究方法与传统语言研究方法不同。它在还原论的基础上, 加入了系统论的研究思想, 即从总体上来观测、分析和解剖语言现象。语言作为人类的产物, 除传统语言学范畴外, 还反映了人类在多方面的特征, 包括心理特征(心理学)、环境特征(整体性)、脑机制(神经网络)等方面。

由此可见, 认知语言学与传统语言学相比, 从一个更加深入和全面的角度来证明或证伪传统语言学的结论。为此, 总结认知语言学的研究方法有如下两个方面(结合本节的内容着重探讨在语义方面的研究方法)。

- ❑ 对传统语言学中一些约定俗成的观点和认识, 认知语言学都设法通过心理学的实验来证明, 并提供数量方法, 尽可能地摒弃研究者的主观判断。
- ❑ 不依靠枚举规则这种确定性的方法来分析语言规律, 而是在丰富的语料资源(语料库)基础上, 为不同的任务建立语言模型, 然后通过模拟神经网络的运行机

制,根据结构关系(线性是序关系,树是父子关系)学习特征,产生模型,再根据模型解析或生成语言。

本书对语义分析持有的主要观点与认知语言学相同,在展开介绍后面的关于深度学习的系列算法之前,我们需要详细介绍认知语言学的一些重要的理论思想,以及由此发展出来的语义结构。首先讲解认知语言学最核心的概念——“象似性”。

8.2.1 象似性原理

19世纪末,美国符号学的创始人皮尔斯(1839—1914年, Peirce)在《皮尔斯:论符号》(以其手稿整理出版的一部符号学研究著作)一书中,首先提出了象似性(Iconicity)原则。那么,什么是象似性呢?皮尔斯所说的“象”是针对符号系统而言的。“象”是指符号系统与客观对象之间的一种相似性的模式。他把“象”分为三小类。第一类是“影像”,符号与对象单纯在某些属性上相似,如象形字。第二类是“图式”,符号与对象在结构或关系上相似,如短语结构。第三类是“隐喻”,即通过此物与彼物的平行性来反映所指物特征的符号。下面会详细讨论这种关系。其实“象”在现代句中代表更广泛的含义,常用来特指某种规律性的语法现象。

为了说明象似性,皮尔斯进一步指出,符号按其与指称对象之间的不同关系可分为象似符(Icon)、指示符(Index)和象征符(Symbol)。当符号与客观对象具有形象的象似特征(如地图、绘画等)时,称为象似符;当符号与客观对象之间存在因果关系(如风向与箭头,症状与疾病,等等)时,称为指示符;当符号与客观对象之间存在约定俗成的联系(如红灯与行人止步,十字架与基督,等等)时,称为象征符。

本书第2章曾经详细介绍过“六书”造字,就汉语而言,象似性的机制已经在造字环节体现得非常充分:所有的象形字本身就是一种象似符;指事字是一种指示符,这与皮尔斯的象似性理论完全吻合,而象征符可以理解为形声字中表语义的形部,这是一个不争的事实。

语言先于文字而产生,语言的发音必然与其所表意义之间存在诸多自然的联系,这种联系一直延续到现在。语音的象似是一种声音上的模仿或复制。这种模仿或复制并不是任意的,而是基于对客观对象的模仿和经验之上的,即使到现代,汉语中的拟声词也是一种语音象似性的鲜明例证。

相较于其他语言,汉语更具相似性的特色,戴浩一曾经说,汉语在更大程度上遵循时间顺序的象似原则。

8.2.2 顺序象似性

顺序是时间在空间上的一种映射。与英语等印欧语言不同,汉语这种缺乏形态变化的语言较少受到句式的影响,在表达语义时,句子是按照动作发生的顺序把各种成分编织到一起的。顺序的象似性(Sequencing Iconicity)可定义为:句法成分的排列顺序映照着它们所表达的实际状态或时间发生的先后顺序,即语言在时间这根轴上单向度地展开,语言结构在顺序上的安排对应于它所表达的概念的次序的安排。

汉语句法的基本序列直接映射了动作发生的顺序。这种情况是现代汉语的一条重要的句法规则。可以从很多语法现象中看出这个规律。在一般情况下,句子的词序与动作发生的顺序是一致的。不同的动作序列,甚至还会导致句子语义的不同。

(1) 他坐公共汽车到这儿。——他到这儿坐公共汽车。

(2) 他在马背上跳。——他跳在马背上。

(3) 张三到图书馆拿书。——张三拿书到图书馆

(4) 她结了婚,生了孩子。——她生了孩子,结了婚。

在短语(组块)成分中,定中关系的短语表现出的顺序象似性:限制性定语→描写性定语。

(1) 两串带露水的新鲜紫葡萄。——限制性:两串;描写性:带露水的新鲜紫

(2) 挂在门前的两个大红灯笼。——限制性:挂在门前的两个;描写性:大红

(3) 信封右角邮票上的彩色图案。——限制性:信封右角邮票上;描写性:彩色

(4) 世界上发展中国家的经济前景。——限制性:世界上;描写性:发展中国家的

限制性定语的顺序按照范围大小顺序排列。描写性定语根据与中心语之间的概念距离远近的顺序排列:大小→新旧→来源→颜色→形状式样→质料→功能。

8.2.3 距离象似性

距离象似性(Proximity Iconicity)。语言成分间的距离象似它们所表达的概念成分间的距离,就叫作距离象似。也有人把它称为“相邻原则”,即:如果两个成分在功能、概念或认知上相互接近,那么在符号层面也相互接近。简而言之,“语符距离象似于概

念距离”。所谓概念之间的接近，按照义素分析法，如果两个概念在语义上具有较多的共同特征，或较为接近的上下位关系，或同属一个概念范畴，那么这两个概念的距离彼此接近。

根据距离象似性原则，概念上距离近的成分总是设法在句法分布形式上彼此靠近。

(1) 动作及其补充成分的距离较之其他的辅助成分更具有象似性。

看完了——*看了完。

吃饱了——*吃了饱。

上述两个例句中，“完”在语义上表示“看”的结果，“饱”在语义上表示“吃”的状态。它们的语义距离明显较“了”更接近于它们所修饰的动词。

(2) “动作发起者”和“动作承受者”与“动作”之间的排列与距离象似性。

① 花猫捉到了老鼠。

② 花猫把老鼠捉到了。

③ 老鼠被花猫捉到了。

④ 那只老鼠，花猫捉到了。

(3) 表示位移动作的成分之间的排列及其距离象似性，位移动作的句法成分之间排列组合的距离远近可以象似性地凸显概念领域事物的距离远近。

① 小王送给小李一本书。

② 小王送一本书给小李。

① 淡化了书转移的中间过程，凸显了书转移的最终结局：送达；②则凸显了书转移的中间过程，淡化了结局。

8.2.4 重叠象似性

重叠象似性，也称为重叠相似动因。归并相同的物体，重复相同的动作，在句法或词法构造上用重叠或重复的形式来表达这些意义。重叠象似性是指：“语言表达形式的重叠（重复）对应于概念领域的重叠（重复）。”

(1) 量词及数量结构的重叠形式。单个量词的重叠式表“每”，如“个个都很好”、“次次都参加”。数量结构的重叠式有时表“逐一”，如“一首一首地唱”、“一口一口地吃”；有时则表动作的多次重复（做状语时，如“一趟一趟地去”）或物体的数量很多（做定语时，如“一串串葡萄”）。共同点：它们都表示相同的事物或动作在量上的迭加或重现，该含义在形式上是用相同成分的叠加或再现来表现的。

(2) 动词重叠式同样含有一种量的观念，即表示动作的量。动词重叠式既可以表示时量短，又可以表示动量小。

① 陪我聊聊天。

② 晚上想去看电视。

③ 你试试这个。

④ 这顶帽子你戴戴看。

(3) 动词形容词重叠 (AABB)。

① 形容词重叠 (AABB)：“大大小小”、“高高低低”、“稀稀疏疏”、“花花绿绿”。

表示某种参差不齐的状态，其所修饰的事物必须多于两个，形式上的重叠反映了概念领域里事物的重叠。

② 动词重叠 (AABB)：“来来往往”、“说说笑笑”、“进进出出”。

表示动作很频繁或相连不间断的意思。

汉语中的象似性绝不仅仅只有上述这些情况。在表达空间结构、领属关系、概念范畴时，汉语都有强烈的象似特征，限于篇幅这里不一一列举。

本章揭示出象似性的原理，如果把语料看作蕴含着各种语言模式的资源，那么语料上的标签所起的作用就是对这些语言模式的一种指示和判别。算法的训练过程是要通过学习这些语言模式，形成一套“象似”的语言模型，而预测过程则是在未标注的新语料上发掘出相应的语言模式。无论是我们前面讲过的概率图模型，还是后面要讲的深度学习模型，这些所有学习算法的本质都要体现出这种“象似”的机制。

“象似性”开端了认知语言学，也可以说是认知科学。一百多年来，随着心理学、脑科学的发展，认知语言学的理论也从这个起点开始不断地发展丰富。接下来讲解有关认知语言学的更多内容。

8.3 意象图式的构成

8.3.1 主观性与焦点

语言的“主观性”(Subjectivity)是指“说话者在说出一段话的同时表明自己对这段话的立场、态度和感情,从而在话语中留下自我的印记”(沈家煊)。

上述抽象的论断让人有点不知所措。下面换一个角度,为了进一步说明“主观性”,先来看一个著名的心理学实验。

在一次国际心理学会议正在举行的时候,突然从外面冲进一个村夫,后面追着一个黑人,手中挥舞着手枪。两人在会场中追逐着,突然“砰”的一声枪响,两人又一起冲出门去。事情发生的时间前后不过20秒钟。在与会者的惊慌情绪尚未平息的时候,会议主席请所有与会者写下他们目击的经过。

结果,在上交的40篇报告中,没有一个人的记载是完全正确的。其中,只有一篇错误率少于20%,有14篇的错误在20%~40%之间,12篇的错误率在40%~50%之间,剩下的错误都在50%以上。

虽然每个人都注意到两人之中有一人是黑人,然而40人中只有4人的报告说黑人是光头,符合事实。其余有的说黑人戴了一顶便帽,有的甚至说他戴了一顶高帽子。关于他的衣服,虽然大多数人都说他穿一件短衣,但有的人说这件短衣是咖啡色的,有的说是红色的,还有人说是条纹的。事实上,黑人穿的是一条白裤、一件黑短衫,系一条大而红的领带。

在心理学中,注意是指心理活动对一定对象的指向和集中。其中,注意的指向性是指人的心理活动总是有选择地朝向一定的刺激物,而同时离开其余刺激物的特点,反映了心理活动的对象和范围。注意的集中性是指人们把心理活动贯注并维持在某一对象上,使心理活动不断地深入下去,反映了心理活动的水平和强度。

从上述定义中,我们可以体会到什么是“主观性”,所谓主观性就是心理活动的选择性。而焦点就是心理活动所选择的客观对象。

上例中,人们都注意到了“黑人”,原因如下:一是因为黑人和在场的多数白人之间肤色有明显差异,容易辨识;二是因为黑人“手中挥舞着手枪,还发出了‘砰’的一声枪响”——焦点,这个场景对与会所有人员的人身安全都构成了威胁。这两个特征足以引起绝大多数人的注意。但向更细节方向考察,人们就不能达成共同的认识。黑人的体

貌、穿着这类特征，对于很多人而言，在事件整个发生的过程中构成不了一致的焦点。

需要说明的是，注意不是一个独立的心理过程。一方面，从客观环境的角度来看，注意与事件发生的场景密切相关。为了更好地理解焦点与场景的关系，我们再看一个例子。

“一辆载着 285 名旅客的火车驶进车站，在这里下去一些人，又上来 85 人；下一站上来 101 人，下去 69 人；再下一站下去 17 人，上来 15 人；再下一站下去 40 人，只上来 8 人；再下一站又下去 99 人，上来 54 人。”

“火车继续往前开，到了下一站……再下一站……再下一站……”

“这趟火车究竟停靠了多少站？”

这个例子是脑筋急转弯的一个题目，乍一看有点无厘头，但从心理学的角度却是焦点误导的典型例证。它通过一个侧面证明了场景、主观性和焦点之间的密切关系。第一段话，说话者暗示了一系列的信息，即火车每停一站，就上来一定的人数、下去一定的人数，整段中通过重复、不断强化这种暗示，使听话者的注意力继续集中在人数的变化上。而焦点就是上、下车人数的具体增减，以及剩余人数。第二段话是第一段的再次重复，进一步加深了强调。其目的是使听话者预测并确信关注的焦点就是最终问题。

从生物学特性来看，来自外界的信息常常是大量的，而不幸的是，人的神经系统高级中枢的加工能力则非常有限，人类必须通过某种过滤机制来对之加以调节，选择其中较少的信息，使其进入高级分析阶段。Broadbent 证明，知觉构成的过滤器一次只允许一个通道的信息通过，即单通道过滤，而不能对不同的事物同时产生不同的知觉。

因为前两段话多次强化（暗示）了焦点，使听话者将注意力都集中在了上、下车的人数上，从而忽略了其他的信息。当真正的问题出现的时候，听话者发现问题虽然比设想的要简单很多，但属于已经被忽略的信息范畴，所以回答不出。

在语言学的句义分析中，由于场景信息的不足常常导致语义模糊。

(1) 咬死了猎人的狗。(太著名了，不解释)

(2) 开刀的是他的父亲。(开刀的医生还是患者)

(3) 这个人谁都不认识。(是谁都不认识他，还是他不认识任何人)

另一方面，从心理的角度来看，人类对焦点的选择与当事人的经验、记忆、兴趣、情感、评判等心理因素密切相关，它是各种心理过程共同影响的结果。不同的动机、性格、经历，以及知识背景等因素都影响了人们对同一场景的注意，因而产生了不同选择

性，它们所构成的焦点当然也不同。

在语言学中，Iwasaki 将这些主观性共分为如下三类。

- 视角化的主观性。说话人的视角 (Perspective) 使说话人处于一个自我、当前时间、当前位置的三维空间中。
- 体验的主观性。说话人的情感 (Affect) 是指说话人对自身感知、感觉、情感和意志的自我意识。
- 认识的主观性。说话人的认识 (Epistemic Modality) 包含了评判、言据、信心的一系列现象，以各种方式限定了说话人的陈述方式。

第一种与场景有关，它直接形成了后面所讲的意象图式；第二种与情感有关，它构成了计算语言学的一个重要分支——情感分析；第三种即我们常说的说话者的立场。

关于第三种情况，由于立场不同导致的语义混淆如下。

- (1) 小李让小张偷了一个钱包。让：致使，还是被动。
- (2) 小李借了小张 100 元钱。借：借出，还是借入。
- (3) 往前倒一下磁带，你就能找到那首歌了。前：时间上的早，还是晚。

由此可见，主观性是人类语言最本质的特征，失去了主观性和焦点，语言还表达什么意义呢？

8.3.2 范畴化：概念的认知

1. 概念：原型与属性

概念在大脑中的产生是一个极其复杂的过程，目前已经可以通过一系列的心理实验来证明。受篇幅所限，这里仅给出一些实验的结果及总结出的规律。从心理学的角度来看，人们对自然现象的概念化（或称为范畴化）经历了如下三个过程。

- 焦点的生成。这源于对外部刺激的选择。在大量能为人的感官系统所感知的刺激中，只有很少的刺激被人的认知系统所选择，这些刺激就称为焦点。
- 鉴别和分类。在获得外部焦点后，人们会对比焦点和存储在记忆中的相关知识（一系列的属性特征），经过对比找到对应的类别，产生对焦点的认知信息（时间、空间、名称等）。注意，焦点事物能够在短时记忆中得到准确的存储，并在长期记忆中更容易保留。

- 命名。所谓命名，就是将外部刺激与大脑中的知识对比，如果发现有相同或相似的类别就为外部刺激赋予对应的名称；如果没有，就设法标识为新的对象，并且为之命名。需要指出的是，人们对焦点事物的感知比非焦点事物要显著，而且焦点的名称也容易被习得。

因此，概念从本质上看是一个心理过程，其存在形式是人的大脑对客观事物共同的、稳定的认知。词汇是概念的命名，即概念的表征。概念所对应的最典型的、出现频率最高的客观事物——焦点，就是概念的原型。原型可以看作概念所对应的客观事物的典型代表。

下面看看如何通过属性来界定一个概念。

图 8.4 所示为美国著名的语言学家 Labov 利用类似杯子的容器图形所做的一个重要的实验。实验的目的是从上述所有给出的容器中找到可以被命名为“杯子”的容器。该图中最左侧的容器为杯子的原型。通过这个实验可以说明概念、原型与属性之间的关系。

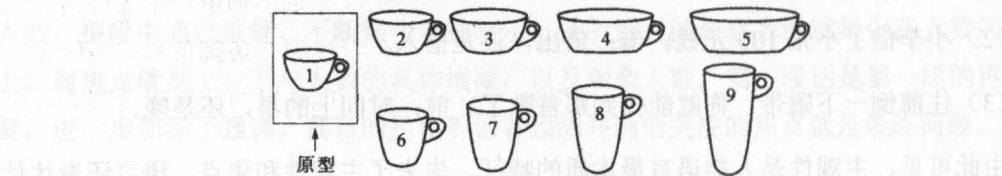


图 8.4 Labov 利用类似杯子的容器图形做实验

现在来考察一下产生这个判断的心理过程。

Labov 将图 8.5 中的曲线定义为“一致性曲线”。该曲线表明图 8.4 中的每个容器与原型对比的相似度。为了更清晰地对比，图 8.5 中还提供了一条与“碗”概念相似度的互补曲线。可以看到，随着容器序号的增加，曲线的相似度值逐渐衰减。

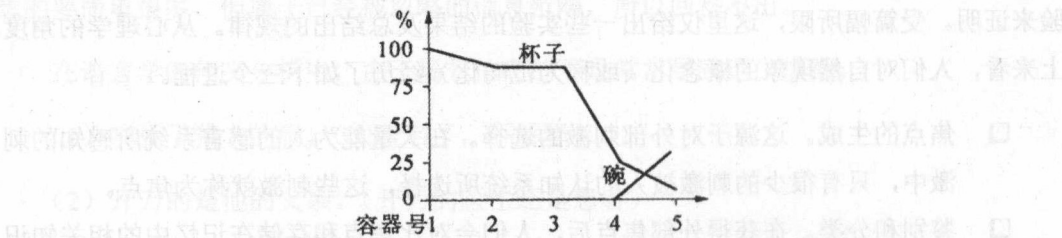


图 8.5 中性语境下的一致性曲线（引自 Labov 1973）

从 4 号开始，“一致性曲线”急剧下降。也就是说，从 4 号往后的各个容器，人们将其看作“杯子”的概率急剧降低。而人们将其认为是“碗”的概率却逐渐提高。

通过 Labov 的实验,总结有关概念、属性形成范畴的基本过程如下。

- 概念在大脑中是以属性集合的方式进行存储的,这些信息主要来自,但不限于视觉、听觉、触觉、嗅觉等感官系统的信息获取,包括:空间上有形状和结构等、时间上有运动和变化等;除此之外还有颜色、材质、情感等特征。注意,任何一个概念的属性都是可以枚举的。
- 概念和范畴是对同一种事物的不同表达。概念侧重于命名和内在结构;范畴侧重于概念所包含对象的范围。
- 范畴的边界是模糊的,相邻两个范畴的边界不能被明确地分开,而是有相互重叠的部分。
- 原型常常位于范畴的中心位置,是实物的标准化或抽象。根据样本与原型的相似度,样本可以划分为若干等级。

2. 范畴与层次

回顾一下第2章的例子。一个人去河边钓鱼,一共钓到一条小鲫鱼和一条鲤鱼。回家后,他会跟人说:今天,我钓到两条鱼。而不会说:今天我钓到一条鲫鱼和一条鲤鱼。更不会说:今天我钓到两种水生动物。这个例子看似简单,反映出的认知行为却是深刻的。它反映出人类大脑组织概念的一种方式——层次。

描述范畴的基本属性最早来源于感官系统对世界的认知。包括视觉、听觉、嗅觉、触觉等。目前来看,这类感官的认知水平虽然水平较低,但是它们形成了人们最初的认知模式。不同概念的大量属性在思维中会被抽象出来,形成更抽象的概念。它们与下层具体的概念形成了层次关系。上层更抽象的概念就称为上位范畴(或上义词),下层较具体的概念就称为下位范畴(下义词)。上例中,鲫鱼和鲤鱼都属鱼类,它们都具有鱼类的共同属性,所以在习惯上更容易用它们共同的上位概念来表达。

但是,属性的抽象并非没有原则。例如,狗、鱼、猫都属于动物,在我们的意识中,狗与鱼和狗与猫相比差异更显著,因为狗的属性集合与猫的属性集合的相似度更大,而与鱼的属性集合的差异度更大。人们对事物的概念会稳定在一个既具体又抽象的平衡点上。这也就是上例所说的“我钓到两条鱼”的主要原因。

因此,分类并不是一个简单的事情。认识这一点比之前的层次结构更重要。笔者曾经花了很长时间设计知识库的分类系统。为此,我们下载了“维基百科”和“百度百科”的全部语料,并参照“维基百科”和“互动百科”分类系统,做了详细的分析。虽然几个百科知识系统各不相同,但是它们的顶层分类体系确实非常相似,基本上都包括地理、

技术、经济、科学、历史、人物、社会、生活、体育、文化、自然十一个大类。为了进一步说明，以生活、地理、技术三个大类为例，其内部结构如图 8.6 所示。

生活	地理	技术
宠物.xml 82 KB	地理理论.xml 46 KB	电信技术.xml 1 KB
健康.xml 289 KB	地图.xml 1 KB	高新技术.xml 1 KB
交通.xml 226 KB	地形地貌.xml 89 KB	航空航天.xml 23 KB
节日.xml 115 KB	各大洲地理.xml 570 KB	互联网.xml 10 KB
居家.xml 137 KB	国家.xml 152 KB	化工能源.xml 11 KB
礼品.xml 1 KB	行政区划.xml 458 KB	计算机技术.xml 103 KB
礼仪.xml 1 KB	旅游.xml 1,030 KB	节能.xml 1 KB
楼盘.xml 1 KB		科技产品.xml 21 KB
旅游.xml 1,030 KB		科学技术.xml 78 KB
情感.xml 1 KB		通信技术.xml 2 KB
日用百货.xml 29 KB		照明.xml 4 KB
时尚.xml 88 KB		
网络.xml 23 KB		
饮食.xml 116 KB		
游戏.xml 34 KB		
娱乐.xml 547 KB		
知识言网.xml 2 KB		

图 8.6 生活、地理、技术三个大类下的子类

如图 8.6 所示，因为类别众多，这里无法给出所有的子类别。以三个类别为例，然后逐层研究。分析的结果如下。

- ❑ 以科学分类法为标准建立的分类系统（生物分类），它们所使用的属性特征很少被用来在生活中描述客观事物，相比之下人们更喜欢通过直觉获得的特征来分类和描述客观事物，即便它们很多与科学分类法相矛盾。例如，“猫”和“狗”的距离较“猫”和“兔狲”在“生活”类中要近很多。它们（“猫”和“狗”）都属于宠物类。
- ❑ 凡是与人们生活越密切相关的事物和概念，其种类就越复杂。一方面这些事物和概念在生活中最常见；另一方面，这些概念在人们的活动中也最易得，其类别维度自然就复杂。最复杂的层次出现在第三层或第四层——“属层次”(Generic Level) 中。以旅游为例，旅游.xml 共有 1MB 多的内容，是三个类别中最大的一个子类，但旅游类别的下位子类仅有三个：中国旅游、外国旅游和旅游知识，几乎所有的内容都集中在这三个子类的下级类别——“属层次”上。世界旅游的子类如图 8.7 所示。

```

<category child="0" id="11392">世界旅游<category child="0" id="18310">各国旅馆<cate
> <category child="0" id="18311">新加坡酒店</category>
</category>
<category child="0" id="23893">澳大利亚观光<category child="0" id="23894">西澳洲郊
</category>
<category child="0" id="33795">马来西亚旅游<category child="0" id="34392">吉隆坡则
</category>
<category child="0" id="34392">吉隆坡购物中心</category>
<category child="0" id="38145">日本旅游<category child="0" id="9734">日本世界遗产
> </category>
> <category child="0" id="31068">日本世界自然遗产</category>
</category>
<category child="0" id="11119">日本公园<category child="0" id="17998">东京公园<
</category>
<category child="0" id="15503">东京旅游<category child="0" id="17998">东京公园<
</category>
</category>
<category child="0" id="65098">巴西旅游<category child="0" id="65100">巴西景点<ca
> <category child="0" id="65124">巴西海岛景点</category>

```

图 8.7 世界旅游的子类

- 一个事物很少仅有一个类别维度，多数事物都有两个以上的类别维度。例如，“旅游”类别既包含在“生活”大类下，也包含在“地理”大类下。再如，“计算机”这个小类，既可以是家用电器、娱乐用品，也可以是高科技产品，还可以是“技术”类别中的一种。可以讲，很难用一种维度来涵盖某种的事物。那种根节点为唯一节点的分类体系不符合人们对事物的认知规律。

图 8.8 所示为人的大脑中的范畴与层次。

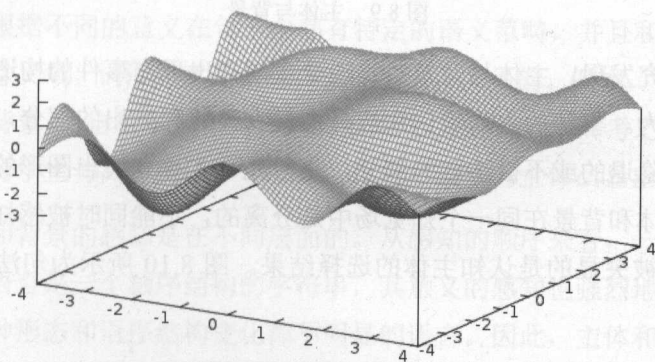


图 8.8 人的大脑中的范畴与层次

图 8.8 形象地展示了人的大脑中的范畴与层次的系统，山峰的地方表示人们对某个范畴拥有的知识更多，其类别的层次就越复杂。而低谷的地方则说明，人们对这类概念所知甚少。

8.3.3 主体与背景

主体 (Figure) — 背景 (Ground) 理论来源于艺术设计的构图技法。构图上的主体所占位置突出、面积较大、刻画细致、色彩鲜艳。而背景主要用于衬托主体, 烘托画面气氛, 刻画尽可能简单。

如图 8.9 所示, 左图中, 在书店 (或图书馆) 看书的女孩, 就是一个典型的主体与背景的布局模式。主体是聚精会神看书的女孩。图片对女孩神态的刻画细致入微, 造型上通过服装的黑白搭配, 突出头部特别是目光的聚精会神、表情专注; 背景是摆满图书的书架, 为了烘托主体特意对其做了模糊处理。左图使观看者在第一时间将目光焦点投射到看书的女孩, 便于明确阐述图片的主题。产生第一印象之后, 再注意被弱化的背景, 使观看者进一步加深对主体的理解。

右图是此类构图的抽象示意图。背景营造了一个时空氛围, 在这个时空中, 通过焦点、颜色、光影的刻画将主体凸显出来。



图 8.9 主体与背景

人们经过研究发现, 主体与背景对比现象也同样出现在事件的构造上, 是人类的一种普遍的认知行为。事件的主体是认知选择中最为显著和突出的部分, 是注意的焦点; 背景是在认知中隐退的或不被注意的部分, 是衬托、支撑和突出图形的认知参照点, 突显程度较小。主体和背景在同一个知觉场中是分离的, 不能同时被感知。主体感知的原则是突显原则, 被突显的是认知主体的选择结果。图 8.10 所示为句法主体与背景的示意图。

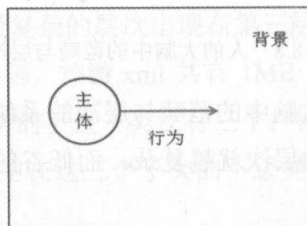


图 8.10 句法主体与背景的示意图

为了便于说明，列举如下几个实例。

- (1) 外商投资企业在改善中国出口商品结构中发挥了显著作用。
- (2) 高考之后，毕业班的学生就自由了。
- (3) 街边放了一辆自行车。

表 8.1 给出上例的详细说明。

表 8.1 主体—背景的认知语言学分析

例 句	分 析
外商投资企业在改善中国出口商品结构中发挥了显著作用	主体：外商投资企业 背景：在改善出口商品结构中 行为：发挥了显著作用
高考之后，毕业班的学生们就自由了	主体：毕业班的学生们 背景：高考之后 行为：自由了
街边放了一辆自行车	存现句 主体：自行车 背景：街边 行为：（被）放了

主体—背景下的语意凸显可以表现为如下几点。

第一，词汇根据不同的意义在句子中都有特定的语义范畴，并且和相关词项构成特定的彼此联系的意义域，词汇的意义就是在域的范围内容凸显的。例如，“街角”构建了一个空间域，形成背景，“自行车”作为背景的主体就凸显出来。“高考之后”构建了一个时间域，毕业班的学生与高考构成一个联系，很自然就作为主体凸显出来。

第二，主体和背景的感知是在不同层面的。从感知的顺序来看，一般先感知主体，后感知背景，但语言是一个顺序结构的字符串，其意义的感知也强烈地受到序列结构的影响。汉语是一种形态和语序结构变化都不明显的语言。因此，主体和背景的感知也受到语序的影响。有关背景的句法成分常常以表示位置和时间的“介词”或“副词”作为标句词构成一个独立的语块。

第三，对于一个合法的句子，主体是必需的，背景则不是。因此，构成主体的各个句法成分是理解语义的关键。构成的背景句法成分是可选的。它是后来形成的语义框架技术的理论基础。在事件语义的使用和识解过程中，主体的意义总是在一定的背景条件

下突显出来的,有时因为上下文的关系,背景在事件中常被省略,但对于独立的句子(不一定是独立的事件),缺乏背景的语句比背景充分的语句更容易造成歧义。

主体—背景理论的一个重要的结论是产生了角色原型的概念。根据句子中不同成分所处的主体与背景的关系,人们建立了“施事 (Agent)”、“受事 (Patient)”、“工具 (Instrument)”、“感事 (Experiencer)”等语义角色,也称为“论元角色”。有关语义框架和语义角色标注的内容,第9章将详细讲解。

主体—背景理论的另一个重要的贡献在于,加深了人们对语义完整性的认识。如前文所说,句子是用来传递语义的完整结构,我们的论证是从指称—陈述分离的角度而言的。从认知语言学的角度来看,一个完整的句子(这里特指单句)构造了一种时空环境,它所表达的主体对象在这个时空中发生了一次行为(或一系列完整的行为:复杂单句的情况)或反映了一种状态。因此,可以把一个句子及其对应的完整语境信息看作独立时空中发生的一个事件。那么,该事件就构成了一个独立的语义空间。在这个语义空间中,句子的主体和背景都独立于它的上下文。在主体—背景理论下的事件概念解决了句子在语义上的模糊问题。

8.3.4 意象图式

有了主体 (Figure) 和背景的概念,很多新的概念就能够清晰地建构出来。在规定的背景下,在主体范畴中,动作的发起者 (Subject) 和动作的承受者 (Object) 经常具有一种模式化的运动关系。在这些关系中,背景和主体都被抽象出来,成为一种心理过程。这就是意象图式的最初来源。

在引入意象图式的概念之前,先介绍“图式”和“意象”两个名词的含义。

“图式”,最初在20世纪20年代,由英国心理学家 F.Barlett 发现。他说:人的记忆能够把各种信息和经验组织成认知结构,形成常规图式,储存在人们的记忆中,新的经验可通过与其对比而被理解。之后,瑞士心理学家皮亚杰运用“图式观”强调认识主要来源于主体 (Subject) 与客体 (Object) 之间的互动,即主体或客体通过自我调节来同化或创立的心理图式。所谓“图式”,就是指人们把经验和信息加工组织成某种常规性的认知结构,并且这种认知结构是通过主体和客体之间的互动得到的,可以较长期地储存在记忆之中。

“意象”常作为心理学的术语,多指一种心理表征。“意象”,是指人在某物不在场时心智中还能想象得出该物的形象,这是在没有外界具体实物刺激输入的情况下,人在心

智中依然能够获得其印象的一种认知能力。

“意象图式”是在人与客观世界感知和互动程序中一种反复出现的、动态性的模式，它为人的经验提供连贯性和结构性的模式。

说得通俗一些，意象图式产生于由一段时间和空间构成的一种场景，场景在时间上是连续的，在空间上具有结构（可以分层）。这个场景中还有主体（我们自身，注意这里与主体一背景中的主体不是一个概念）和客体（外部事物）。主体和客体的实际互动过程曾经反复地出现在人的社会实践中，人们已经对此形成了记忆，并找到了规律。当同类场景再次出现时，人们能够自然地激活（或回忆）场景中的动作模式、完成互动，或者可以通过思考，改进原有的动作模式。

意象图式产生于人类的具体经验，即来源于人体在外部空间世界中的活动，因此具有体验性。在结构上，它是许多具有一些共同特点的活动的“骨架”，用来组织人类的经验，把看似无关的经验联系起来。因此，意象图式是一种抽象性的思维结构，人类可以把它映射到抽象概念中。它是人类的抽象概念的一个重要来源（第二个是隐喻）。

常用的意象图式是有限的。Johnson（1987 年）共谈到如下 20 多个意象图式，包括：CONTAINER、BALANCE、COMPULSION、BLOCKAGE、COUNTERFORCE、RESTRAINT、REMOVAL 等（按照惯例意象图式都使用大写字母）。而 Lakoff 将其概括为 7 类意象图式：容器、起点—路径—目标、连接、部分—整体、中心—边缘、上下、前后。另外，还有 4 种基本图式：力、运动、平衡和对称。

本文摘要其中的 8 种如图 8.11 所示。

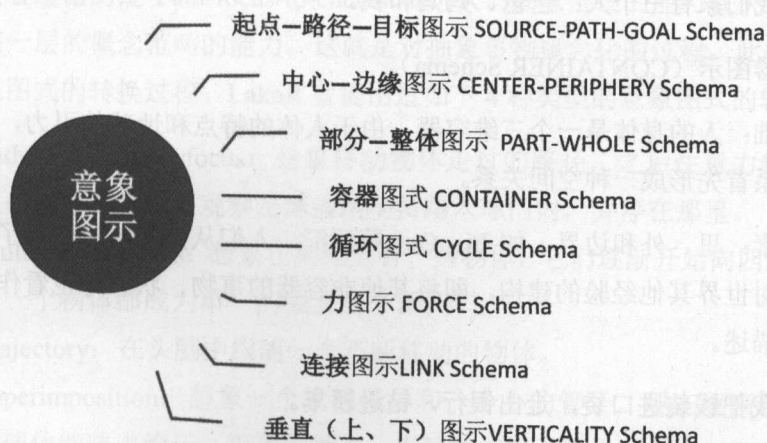


图 8.11 意象图式的类型

□ 起点—路径—目标图示 (SOURCE-PATH-GOAL Schema)。

生理基础：当物体从一个地点移到另一个地点时，一定有起点、终点和路径。

构成要素：起点、终点、路径和方向。

目的被看成终点，到达目的是“到达终点”。

例句：小张从北京出发，驱车向南一千六百多公里，终于到了上海。

□ 中心—边缘图示 (CENTER-PERIPHERY Schema)。

生理基础：人体具有中心（躯体和内脏器官）和边缘（手指、脚趾、头发等）。同样，树和植物具有树干、树枝和树叶。

构成要素：实体、边缘。

中心是重要的，边缘是不重要的，边缘依赖中心而存在。

例句：夕阳西下，越过远山。

□ 部分—整体图示 (The PART-WHOLE Schema)。

生理基础：人本身及其他物体是由部分组成的整体。

构成要素：整体、部分和构造。

在此图示中，只有部分存在于同一结构中才构成整体。人的经验将家庭及其他社会团体视为由部分组成的整体，如离婚被视为“解体”。

例句：我们家有一个人：爸爸、妈妈和我。

□ 容器图示 (CONTAINER Schema)。

生理基础：人的身体是一个三维容器，由于人体的特点和地球的引力，人与外部世界的这种关系首先形成一种空间关系。

构成要素：里、外和边界。如 In、Out 等词汇，人们从空间结构获得了这种图示，又将它用于对世界其他经验的建构，即将其他非容器的事物、状态等也看作容器，并依此来认知和描述。

例句：我把钱装进口袋，走出银行，钻进轿车。

□ 连接图示 (LINK Schema)。

生理基础：人的第一连接物是脐带。

构成要素：两个实体和一个连接关系。

社交和人际关系被看成一种连接关系。

例句：奴隶制被视作一种“桎梏”。

□ 循环图式 (CYCLE Schema)。

构成要素：起点、不受阻碍的事件进程，以及最终回到起点。

路径轨迹上有一个从高峰到低谷的结构。

□ 力图式 (FORCE Schema)。

它涉及物理或隐喻性的因果互动，包括如下要素：力的来源和目标，方向和强度，以及因果序列的运动路径。

力图式的种类包括：吸引力图式、平衡图式、阻塞图式、强迫图式、反作用力图式、导流图式、启用图式、约束移除架构等。

□ 垂直（上、下）图式 (VERTICALITY Schema)。

涉及“上”和“下”的关系。类似范畴中的上下位关系。

在有关意象图式的所有研究中，与计算语言学密切相关的是意象图式的转换概念。意象图式的转换 (Image Schema Transformations) 是指意象图式进行概念化的过程中注意的焦点可以发生转换。例如，当一个人的注意力集中在一个在草地上运动着的高尔夫球时，他利用的是动态的路径意象图式，一旦该球停止，他的注意力就会停在球上。因此，观察者遵循的是 Path-focus-to-end-focus。人们抽象推理的能力取决于把感知范畴映射到更高一层的概念范畴的能力。这就是对抽象事物概念化的过程。此概念化的过程包含了意象图式的转换过程。Lakoff 曾提出过如下 4 种类型的意象图式的转换。

- Path-focus-to-end-focus: 想象移动物体走过的路径，之后注意力集中在物体停止的地方。例如，观察足球被运动员踢入球门内，并停在那里。
- Multiplex-to-Mass: 想象在某地点有一些物体，它们逐渐开始向四周移动，直到每一个物体都成为单一的独立的个体。
- Trajectory: 在头脑中跟随一个不断移动的物体。
- Superimposition: 想象一个大的球体和一个小的管子。现在逐渐把管子变大直到球体能装进管子，再逐渐缩小，直到能把管子装进球体。

我们日常生活中的无数经历大都遵循以上意象图式的转变。例如，当你看到一个水

果摊时，决定过去买水果。你从原来的站立处走到水果摊停下挑水果，遵循了 Path-focus-to-end-focus 的转换。你把挑选的水果从小贩的筐里拿出，遵循了 Multiplex-to-Mass 的转换。你首先挑选了一些樱桃，之后是苹果。苹果放在樱桃的上面，这就是 Superimposition。后来，城管人员来了，小贩开始跑。你的视线追随着小贩逐渐跑向远方，这就是 Trajectory。

这一发现直接导致了后来框架语义学的诞生。框架语义学中有 5 个典型的事件框架类型：位移事件框架、致使事件框架、循环事件框架、参与者互动事件框架和相互关系事件框架。下面从一个实例来看一下位移事件框架的简单结构。

例句：1909 年 7 月 26 日 Louis Bleriot 从 Les Baraques 到 Dover 飞越英吉利海峡。

FIGURE主体	MOTION位移	PATH路径	GROUND背景
Louis Bleriot	飞	越	英吉利海峡

8.3.5 社交中的图式

由最基本的意象图式可以构成大量的其他图式，生活中常见的认知图式如下。

(1) 事实和概念图式 (Fact-and-Concept Schemas) 是关于事实的一般知识图式。虽然具体到每个事物个体的形状、颜色及花纹都是不同的，但它们在人们头脑中关于事物的抽象概念——图式是相近的。

(2) 个人图式 (Person Schemas) 是关于不同类型的人的知识，包括人格特征。在社会交往中，人们常常会不自觉地对人进行评判。比如，“善良”、“诚实”、“内向”等。这是因为人们头脑中预存了这些类型的图式，当某个人符合此图式时，就会把这些概念套用到此人身上。有时候个人图式还会形成刻板印象。例如，“上海人”、“南方人”、“北方人”等概念。

(3) 自我图式 (Self Schemas) 是人们对自我的认识，以区别于他人。它是自我概念的重要组成部分。自我概念是人们成长过程中形成的对自我的认识和判定，它和自我预期 (Self-Fulfilling Prophecy) 紧密相连。自我图式包括：“自信”、“独立”、“敏感”等各种维度。人一旦形成一定的“自我图式”，就会用此图式来理解和解释自己在日常生活中的行为表现。比如，为了显示自己的“独立”个性，个人会在发表意见时标新立异，并谢绝他人的帮助，而没有这种自我图式的人则不在乎自己在这方面的表现。

(4) 角色图式 (Role Schemas) 是指对在社会中或在特定情况下具有特定身份角色的人的行为的认识，这种角色图式会产生特定的角色期待。比如，有人认为妻子应该是贤妻良母，既要侍奉丈夫、料理家务，又要孝敬公婆、养育儿女；有人则认为妻子是比

翼双飞的人生伴侣,既是闺中知己,又是事业伙伴。再如对教授的看法,不仅要教好书,还要搞科研;不仅能教好一门课,还应该学识渊博、为人师表,等等。角色图式也和所谓的“刻板印象”有关系,比如,一般认为女人是情绪化和软心肠的,亚洲人比较勤劳,等等。

(5) 情境图式(Context Schemas)是对社会交际的情境场合及相应的适当行为的认识。情境图式帮助人们识别环境,并采取相应的适当行动来实现目标。不同文化中社会交往情景存在差异。比如,同样是葬礼,中西方差别也比较大。西方一般是去教堂参加葬礼,着黑色服装、送鲜花。而中国一般是去死者家中看望、守灵、着白衣、送钱物,参加出丧和随后的宴请等。再比如,发生交通事故后,日本人除付保险赔偿外,肇事者往往要去医院看望伤者,但美国人却没有这一习惯。

(6) 程序图式(Procedural Schemas)也可以称为草案(Script),和情景图式相连,是对经常发生的事件的有序组织的认识,包括采取恰当步骤和行为规则。比如,有去医院就诊经验的人,对如何挂号、诊治、检查和缴费的过程比较熟悉,就比较能争取主动;再如,中国学生对如何申请学校和申请奖学金不太熟悉,感觉比较困难和麻烦。此外,还有各种社会交往程序,如登门拜访,出席各种活动,处理企业中上下级的人事关系,等等。

(7) 策略图式(Strategic Schemas)是对解决问题的策略办法的认识。对情景的识别也会影响到人们对解决问题的策略办法的选择。比如在中国,当个人遇到思想或情绪问题时,常常求助于朋友的劝解和帮助,而在美国可能主要求助于心理医生。对策略的选择往往与人们对某类问题的熟悉程度有关。比如,医院里经常处理急救情况的大夫比没有急救经验的人更善于找到解决办法。策略图式也常常和克服各种条件限制联系在一起,如消防队员要具备争取时间和应付各种意外的知识,登山队员也要具备各种野外生存的知识 and 能力。所以,策略图式往往和专业相联系。

(8) 情感图式(Emotion Schemas)是对愤怒、恐惧、嫉妒、孤独等情感的认识,它们来自个人的生活经历并储存在长期记忆中,而且会和其他图式相联系、相伴随。虽然图式主要是一种认知结构,但研究表明它们也往往和特定情绪相联。比如,一见到牙医就会感到紧张或恐惧等,曾受性暴力袭击的纯真少女可能终其一生都对性持负面情绪,等等。这些和一定图式相联系的情感反映在社会交际中也有很大作用。

各种社会交往图式并不是孤立的,而是相互联系、共同作用的。它们共同构成相互联系的图式塔。智能研究的美国社会心理学家特纳在人工智能研究中表述了计算机对人类行为的模拟过程:

首先，在特定的情况下，我们尽力识别是否认识这一情况。这就需要从记忆中追寻一个甚至几个情景图式（Context Schemas）。然后，当类似于当前情况的情景图式找到之后，该图式会随即提出一个要实现的目标；当目标确定以后，又会进一步寻求完成这一目标的策略图式，而这种策略图式不仅要和情景图式相关，而且和自我图式、角色图式及其他图式相关联。当策略图式选定之后，程序图式（Procedural Schemas）将会提供一系列的行为步骤指导人们的行动。在采取行动的过程中，人们又会进一步判断情况，选择情景图式，从而在更具体的情况下依据与之更相应的图式来行动。当然，这只是理想的理论模式。实际上，在这一过程中，人们往往会进一步修正原有图式、建立新的图式。因为缺乏特定图式而造成的行为失当和达不到既定目标也是常有的事。

由此可见，在构建自然语言的应用系统中，这些图式及对应的语义框架构成了应用系统的功能核心。

8.3.6 完形：压缩与省略

如图 8.12 所示，完形同样来源于心理学的一个实验。一排几何图形，画的是正方形及其一些变形。图 8.12（a）仍旧是原型。图 8.12（b）有一个缺口，图 8.12（c）有一处凹陷……这种实验描述称为完形心理学。我们的目标是要找到所有样本中哪个最接近于原型，并且填充或修正有差异的部分。

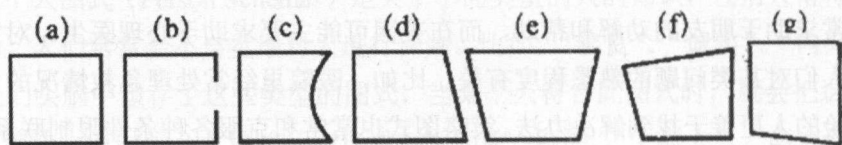


图 8.12 完形的图示

很明显，相比其他的形状，图 8.12（b）最接近于原型。需要填充的部分仅仅是将缺口用直线连接起来即可。完形的现象同样出现在语言学中。汉语中有大量的省略现象。例如，省略主语称为主语脱落；疑问句的答案形式都是对原有句子的某种省略，等等。

完形可以看作原型范畴的一种缺失或变种，这个变种省略或改变了原型的部分结构，但在整体上与原型有着密切的联系。两者之间应该是同构的，至少主体结构是相似的。所以说，完形一定是对原型的完形。完形提供给人类这样一种能力，通过观测事物的某些部分或参照点，即可通过部分来推断整体的原型。通过完形，人类的认知能力大大提高了。

语义事件间的关系与整合就是完形机制在实践中的一种应用。在实际语篇中，句子的结构常出现复杂的复合变化。在汉语中，可以根据句子的结构划分为单句和复句两种类型。复句由两个或两个以上在意义上和结构上有密切联系的分句构成。分句间一般用逗号、分号隔开，用句号、间号或叹号表示结束。分句形似单句，但不独立，它依存于复句而存在。

从语义的角度来看，复句中的每个分句都代表一个事件。因为在一个复句中存在多个事件，我们使用认知语言学中的专门术语——输入空间（Input Space）来表示。也就是说，复句中的每个分句所代表的事件都是一个输入空间。整个复句就构成了一个整合空间（Blended Space）。

复句的认知模式如图 8.13 所示。

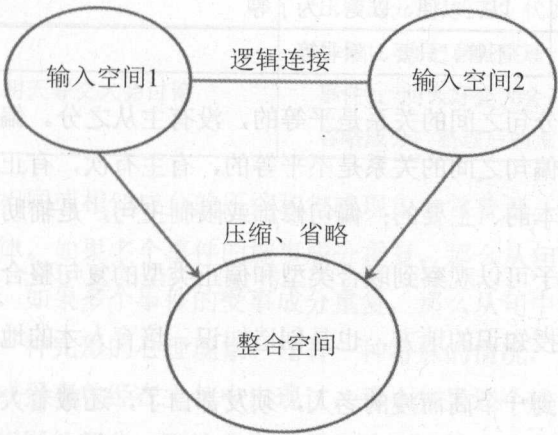


图 8.13 复句的认知模式

如图 8.13 所示，该模型展示了复句的认知构成。每个输入空间代表复句中的一个分句。它们使用某种方式进行连接。所有的分句事件经过连接之后，将相同的背景元素进行压缩，在汉语中常表示为重复成分的省略，变成一个复句结构。此时每个分句都保留了主体结构，压缩了相同的背景结构。

表 8.2 所示为复句中分句的连接类型与关系。

表 8.2 复句中分句的连接类型与关系

类 型	连接类型	连 接 词
联合类型	并列关系	和、跟、与、既、同、及、而、况、况且、何况、乃至等
	比较关系	像、好比、如同、似乎、等于；不如、不及；与其……不如……、若……则……、虽然……可是……，等等

续表

类 型	连接类型	连 接 词
联合类型	选择关系	或、或者、还是、亦、非……即……、不是……就是……，等等
	承接关系	则、乃、就、而、便、于是、然后、至于、说到、此外、像、如、一般、比方、接着等
	递进关系	不但、不仅、而且、何况、并、且等
偏正类型	转折关系	却、虽然、但是、然而、而、偏偏、只是、不过、至于、致、不料、岂知等
	让步关系	虽然、固然、尽管、纵然、即使等
	假设关系	若、如果、若是、假如、只要、除非、假使、倘若、即使、假若、要是、譬如等
	因果关系	原来、因为、由于、以便、因此、所以、是故、以致等
	目的关系	以、以便、以免、为了等
	条件关系	不管、只要、除非等

联合复句的各个分句之间的关系是平等的，没有主从之分。偏正复句由正句和偏句两部分组成。正句与偏句之间的关系是不平等的，有主有次，有正有偏。正句承担了复句的基本意思，是基本的、主要的；偏句修饰或限制主句，是辅助的、次要的。

通过如下几个例子可以观察到联合类型和偏正类型的复句整合。

- (1) 学校既是传授知识的地方，也是创造知识、培育人才的地方。
- (2) 这次，他遇到一个高而瘦的老人，须发都白了，还戴着大眼镜。
- (3) 教室里的灯亮了，我夹着书走进去。
- (4) 一种是教条主义，一种是经验主义，两者都是主观主义。
- (5) 你把意见整理一下，明天好交大会讨论。

表 8.3 给出上例的详细说明。

表 8.3 复句中分句的连接类型与关系

例 句	分 析
学校既是传授知识的地方，也是创造知识、培育人才的地方	事件1：学校是传授知识的地方。 事件2：学校是创造知识的地方。 事件3：学校是培育人才的地方。 省略成分：学校 压缩成分：地方

续表

例 句	分 析
这次，他遇到一个高而瘦的老人，须发都白了，还戴着大眼镜	事件1：这次，他遇到一个高而瘦的老人。 事件2：老人须发都白了。 事件3：老人还戴着大眼镜。 省略成分：老人
教室里的灯亮了，我夹着书走进去	事件1：教室里的灯亮了。 事件2：我夹着书走进教室。 省略成分：教室
一种是教条主义，一种是经验主义，两者都是主观主义	事件1：一种是教条主义。 事件2：一种是经验主义。 事件3：教条主义和经验主义都是主观主义。 压缩成分：“两者”代表教条主义、经验主义
你把意见整理一下，明天好交大会讨论	事件1：你把意见整理一下。 事件2：明天好交大会讨论。 省略成分：整理后的意见

汉语的复句中，相同或相似成分的压缩和省略现象非常普遍。对于省略现象，可以简单地总结出如下规律。如果多个事件的施事成分重复，那么从句中相同结构的施事成分就可以省略。同样，如果多个事件的受事成分重复，那么从句中的受事成分就可以省略。这种情况类似于一种完形的心理现象。还有一种特殊的情况，主句和从句的结构不同，如果从句的施事或受事曾经在主句中出现过，那么如果这个成分是句子的主体，则从句可以省略与主句相同的部分，不论是它出现在施事还是受事的位置。

压缩现象始终是语言解析的一个困难。表 8.3 中给出的压缩一般是对前面出现的事件或成分的压缩。其实，更广义的压缩可以看作一种指代消解的情况，有关这部分的讲解，本书会专辟章节，详细论述。

8.4 隐喻与转喻

隐喻 (Metaphor) 与转喻 (Metonymy) 在传统语言学上都被作为一种修辞手段来研究。修辞是为了使语言更加生动、更富表现力、易于理解，但隐喻和转喻显然不仅仅局限于修辞的功能。在进入 20 世纪以后，I.A.理查兹系统地对隐喻认知特征进行研究，发现隐喻与人类对世界的认识及人类社会的发展密切相关。在其 1936 年完成的《修辞哲学》

中，理查兹对隐喻的认识产生了质的变化：隐喻与思想有关且无处不在；人类对世界的感受是隐喻的。这使隐喻机制脱离了语言的范畴，纳入到思维和认知的领域。

8.4.1 隐喻的结构

1. 隐喻：基于相似

隐喻的产生源于两个事物的“相似”和“对比”，即在心理上直接的“象似性”。作为一种修辞手段，隐喻由“本体”(Tenor)，也称为“话题”(Topic)，以及“喻体”(Vehicle)和“喻底”(Ground)三部分组成。其中，喻底是指两个不同本体与喻体之间的相似性和共同点。将本体与喻体两者进行直接比较，表达方法为：A 是 B，就是把未知的对象转换成已知的对象，借助现有的描述符与未知对象之间的相似性来描写对象。

如表 8.4 所示，传统上一个隐喻结构的显性部分包括三个部分：本体、喻体和比喻词。其中传达的信息一个隐含的部分——喻底，也就是上文说的两者相似的特征。喻底隐含在本体和喻体之间，而被隐藏起来，所以称为隐喻。隐喻是不可以被直接解析的，必须依靠语用推理才能理解隐喻修辞的含义。

表 8.4 隐喻的结构

本 体	比 喻 词	喻 体	喻 底
太阳	是	天空的眼睛	明亮、形状
时间	是	金钱	重要性
人生	像	旅途	过程

2. 转喻：基于邻近

转喻是指当 A 事物同 B 事物有密切关系时，可以利用这种关系，以 B 事物的名称来取代 A 事物，这样的一种修辞手段。转喻的重点不是在“相似”，而是在“邻近”。

(1) 概念转喻。

- 部分映射整体。

例句：退休三年，到系里去见到的尽是新面孔。

- 外形特征映射本体。

例句：去把大头叫来。

- 附属物特征映射本体。

例句：前面走来两个红领巾。

□ 专指映射泛指（个体激活全体）。

例句：我们中国会有自己的哥白尼、居里夫人、爱因斯坦。

例句：不拿群众一针一线。

（2）谓词转喻。

因果激活：弦外之音、旁敲侧击。

例句：他们已经熄灯了（睡觉）。

例句：那家商店关门了（倒闭）。

例句：老佛爷和万岁爷谁走在前面呢（死）。

从上述实例中可以看出，转喻较隐喻在认知上更为深刻。它揭示出如下两种重要的认知原则。

- 凸显原则。在认知过程中，人们往往更加容易注意到对象的典型特征，而转喻是使用典型特征来代替对象本身，从而识别出对象本身的。
- 关联原则。转喻作为一种认知现象，激活了对象的特征或对象周边的相关概念，换句话说，转喻是不可以被直接解析的，必须依靠语用推理才能理解转喻的含义。

8.4.2 隐喻的认知本质

2.2.3 节“会意”造字法的内容就涉及隐喻的问题。本节所指的隐喻（包括转喻）不仅是指文学中的修辞，也是一种认知的工具。Lakoff 和 Johnson（1980）站在认知的角度从功能和结构两个方面对隐喻的概念加以解释。他们认为从功能上看，“隐喻的本质就是用一种事物去理解和体验另一种事物”。其中，最重要的词是“理解”和“体验”，隐喻被引入了认知学的范畴。“隐喻不仅仅是语言的问题，它更是思维的问题。各种思维活动，不管它是关于情感的、关于社会的，还是关于人格的、关于语言的或者关于生与死的，都与隐喻相关。隐喻对于想象和对于理性来说都是不可或缺的。”

Lakoff 和 Johnson 在充分研究隐喻的心理学和认知学意义之后，提出了隐喻认知模型，如图 8.14 所示。

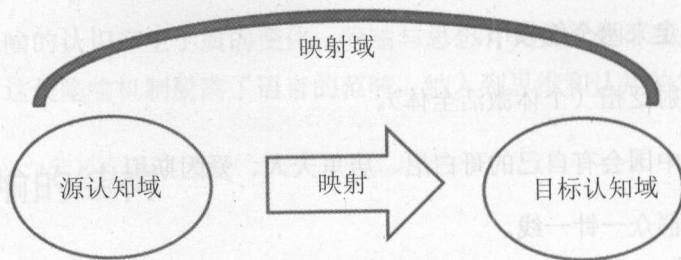


图 8.14 隐喻认知模型

如图 8.14 所示，源认知域是指被隐喻的认知范畴（概念、谓词），类似上文所说的本体，如“太阳”、“时间”、“人生”。目标认知域相当于“喻体”，如“眼睛”、“金钱”、“旅途”。映射域相当于上文所说的“喻底”，这里是指映射的范围，一般是两个概念属性中的共性特征。作为认知的隐喻除“相似”或“对比”之外，还具有更复杂的内在关系，这就是“映射”。“映射”在这里被理解为一种转移，意思是说隐喻把源概念的结构转移到了目标概念上。这是一个重要的过程。这意味着，隐喻不仅仅是一种利用语言手段表达思想、在风格上增加魅力的修辞方法，还是一种对事物进行思维的方法。隐喻所转移的不仅是各个概念的内在属性，还是对概念的整个认知结构、内部关系和逻辑。

因此，隐喻可以用某一领域的经验来说明或理解另一领域的经验，也可以用旧的知识来解释和说明新的知识。通过在两个不同性质的概念间建立联系，它使认知的范围由熟悉的领域向未知的领域扩展、传递。这使它处于人类独有的认知能力的核心部位，负责意义的产出、传送和加工。

隐喻从认知功能角度出发可以分为结构隐喻、方位隐喻和实体隐喻三大类。

(1) 结构隐喻 (Structural Metaphor)，指以一种概念的结构来构造另一种概念，使两种概念相叠加，将谈论一种概念的各方面的词语用于谈论另一种概念，从而产生的一词多义现象。

(2) 方位隐喻 (Orientational Metaphor)，指参照空间方位（往往来自人类对基本空间概念的认知，如上一下、前一后、中心一边缘等）而建立起来的一系列隐喻概念。即人们将其他抽象的概念（如时间、数量、社会地位等）映射到具体的方位概念上，形成用表示方位的词语来表达抽象概念的语言。例如，英语中根据“HAPPY IS UP; SAD IS DOWN”这一概念演变来的“I am feeling down”、“He is in high spirits”等。汉语中“人们的收入水平在不断提高”这种说法就是来自“MORE IS UP; LESS IS DOWN”这个概念的。

(3) 实体隐喻 (Ontological Metaphor)，指人们将抽象和模糊的思想、感情、心理活

动、事件、状态等无形的概念看作具体的、有形的实体，因而可以对其进行谈论、量化、识别其特征和原因等。例如，“The work needs more patience”和“My fear of tiger causes my breakdown”就很好地说明了这个问题。容器隐喻（Container Metaphor）在实体隐喻中最具有代表性。此理论认为人本身可以被看成一个容器，有内外之分。将这个概念投射到人体之外的其他事物上，如房子、地区、田野，甚至把一些无形的、抽象的事件、行为、活动、状态也看成一个容器。例如，“The car is coming into view”和“I’m in trouble now”，等等。

如果说隐喻的修辞学目标是为了传递一种关于情感的体验，那么在认知领域中，它更广泛地出现在将具象概念特征投射到抽象概念上，用来解释和说明抽象概念的作用。

8.4.3 隐喻计算的系统架构

除作为丰富语言的手段，转喻和隐喻都是基于人的大脑中原有的思维方式和基本经验，映射出新的概念的认知过程，但也都不是任意的，可以被规约化。无论是隐喻还是转喻，喻体常常是隐含的。因此，二者都无法直接传达语义，而需要有一个语用推理的过程。这就出现了隐喻计算的问题。

隐喻计算曾经是自然语言处理中棘手的问题之一。国外流行的隐喻计算系统大体可以分为如下两类。

第一类，主要以规则推导为主。根据其设计思想的不同还可细分为如下内容。

- 优选语义思想。这方面的主要代表是 Fass (1988) 提出的可以处理隐喻、转喻、字面义、反常表达的隐喻理解模型 Met 5 系统。
- 基于实例的思想。Martin 利用概念之间的映射关系，提出了识别和解释常规隐喻的 MIDAS 系统。
- 隐喻理解的逻辑推理模型。例如，Gentner, Falkenhainer 的结构映射引擎 (Structure-Mapping Engine) (SME), Holyoak & Thagard 的 ACME 隐喻分析模型。
- 基于隐喻凸显思想、基于语义标记思想的隐喻理解模型以及 Veale 的 Sapper 模型。

第二类，主要以统计技术为手段，基于大规模语料库提取的隐喻分析模型。

这方面的典型代表是 CorMet 模型。Mason (2002) 利用大规模语料动态提取优先选择参数来识别特定领域的隐喻表达。CorMet 系统的出现表明了基于大规模语料的隐喻识别方法正悄然兴起。

国外曾经流行的基于优先语义的思想、基于实例的思想、基于理解的逻辑推理模型都受到语言知识资源不足的限制,在现有的自然语言处理技术水平上,还很难为这三种方法提供比较完备的知识资源。所以,当今隐喻识别的主流技术仍然是基于大规模真实语料库的统计。

汉语中的隐喻现象无处不在,分类标准包括词类和语法单位两大类。按词类分为:名词性隐喻、动词性隐喻、形容词性隐喻等。按语法单位分为:构词隐喻、词语级隐喻、短语级隐喻、句子级隐喻(女人是水)、篇章级隐喻等。

名词性隐喻和动词性隐喻在隐喻的用法中具有代表性和典型性,目前隐喻研究主要以名词性与动词性的短语级隐喻、句子级隐喻为主。

- 源域 n 和目标域 n 接续出现,映现到语言的构词层面,如柳叶眉、杏核眼、蜂窝煤。
- 源域。可以单独出现,映现到语言的词汇层面,源域在词典中有隐喻义项,如风波、风云、摇篮、里程碑。有的也可以和目标域共现,如金融风暴、革命风波。
- 目标域 n 单独出现,映现到语言的词汇层面,如 IT 术语——病毒、菜单、笔记本。
- 源域 n 和目标域 n 非接续出现,映现到语言的短语层面,如知识的海洋、历史的车轮。
- 源域 n 和目标域 n 非接续出现,映现到语言的句子层面。例如,女人是老虎、大地是母亲。
- 目标域 n 缺省,由源域 n 和搭配词所构成的“ $n+n$ ”隐喻模式。例如,“祖国的花朵”,“花朵”的目标域实际上是指“孩子”,而不是“祖国”。

——《汉语名词短语隐喻识别研究》

但是,汉语隐喻计算在国内的研究不多,还处于起步阶段。本文中有关的隐喻计算架构和相关统计数据均参考了国内为数不多的几篇论文。其中,以王治敏的《汉语名词短语隐喻识别研究》一书最为系统、全面地阐述了隐喻计算的过程,并且给出了较为可信的实验结果。

在隐喻知识资源建设方面,北京大学建立了一个小规模隐喻标注语料库,隐喻知识库的建设采取半手工、半机器学习的整理方式,目前已经标注了4个月的《人民日报》语料库,规模不算很大,源域词汇共计95个,目标域词语还未收录完全,标注选择的隐喻方式为“ $n+n$ ”结构的不同层级的隐喻表达。表8.5所示为1998年1~4月《人民日报》的“ $n+n$ ”隐喻分布统计数据。

图 8.15 中的内容说明如下。

- (1) 提取含有关键词的文本。
- (2) 经过分词标注。
- (3) 人工标注形成隐喻标注语料库。
- (4) 通过实例方法、贝叶斯方法、最大熵方法的识别实验结果进行比较，最终确定单个隐喻识别的最佳模型。
- (5) 在单个隐喻识别的最佳模型基础上继续进行实验，建立“n+n”模式识别模型。
- (6) 通过辅助特征、规则提取进一步提高识别效果。
- (7) 在单个隐喻识别的最佳模型基础上进一步识别“n+n”模式。
- (8) 引入 CCD 建立概念相似度推理，提高识别效果。
- (9) 引入名词隐喻知识库的部分属性信息提高识别效果。
- (10) 加工出来的隐喻文本一方面可以为机器翻译和信息检索等实用目标服务；另一方面也可以为建造隐喻知识库服务。
- (11) 识别出来的隐喻文本经过人工校对后自动提取目标域。
- (12) 人工消歧确定[目标域]的概念节点，建立[源域]和[目标域]的隐喻连接。
- (13) 人工消歧通过工具辅助完成。

——《汉语名词短语隐喻识别研究》

从目前的情况来看，由于 word2Vec 对词向量的表达日臻成熟，越来越多的基于知识库的词汇相似度的计算都使用 word2Vec 的词向量来代替，这使得该模型获得了进一步的简化，但在模型构建初期，人工标注工作是不可避免的过程。因此，目前基于隐喻的语料库规模都不大。

8.4.4 隐喻计算的实现

隐喻计算整体上尚处于实验阶段，受限于语料。本节的结果部分参考了《汉语名词短语隐喻识别研究》的相关内容，为了突出重点，这里做了删减。

通过研究发现，一个源域（源认知域）一般只映射有限个目标域（目标认知域），这

说明隐喻中源域对目标域的选择是有限的。如果能从语料中尽可能多地收集这种映射表达,再找出映射背后的规律性,从而形成模型所需要的特征约束,那么如果将其应用到大规模的语料上,可以对未来出现在不同领域语料中的隐喻映射做出很好的预测。

汉语中“名词+名词”(n+n)一类的隐喻现象最多,语法形式上,隐喻形式和非隐喻形式无明显差别,更无规律可循,识别起来难度也最大。因此,研究的目标主要针对“名词+名词”(n+n)形式的隐喻进行识别研究。

选择的语料来自《人民日报》,方法上参照概率图模型,将隐喻识别转化为隐喻义和字面义的二值分类问题,通过标注一定规模的隐喻语料库训练分类器参数,进而利用训练好的分类器对开放语料进行自动隐喻识别。

隐喻计算实现利用统计机器学习的算法策略,分别用最大熵算法、CRF 算法、SVM 算法进行训练。

1. 选择语料

实验选取了 1998 年 1~4 月《人民日报》标注语料作为训练样本,测试语料主要抽测了 2001 年《人民日报》标注语料的部分样本。与单个词语的隐喻标注相比,n+n 结构中包含了更多的隐喻表达类型,无法采用隐喻和非隐喻的标注方法,因此使用“< >”在文本中标注。即首先从 1998 年(1~4 月语料)抽取全部的“n+的+n”、“n+n”的组合形式,然后从这些组合中找出隐喻的用例。

关于测试语料的抽取。测试语料文本抽测了 2001 年《人民日报》2 000 个句子(仅包含 95 个源域词语)。95 个源域词语如表 8.6 所示。

表 8.6 95 个源域词语

主线	种子	支柱	钥匙	雨露	阴云	摇篮	阳光	烟水	芽
漩涡	序幕	星	峡谷	舞台	微波	土壤	隧道	曙色	曙光
食粮	森林	热浪	桥梁	旗帜	喷泉	纽带	泥潭	泥坑	命脉
门槛	脉络	脉搏	绿洲	路线	龙头	灵魂	浪潮	精髓	脚步
枷锁	脊梁	激流	基石	火炬	火花	火	灰尘	花	洪流
河流	河	航船	海洋	瑰宝	光线	光辉	光	痼疾	根
高峰	杠杆	甘泉	风浪	常青树	风帆	风潮	风波	风暴	动脉
殿堂	堤坝	低谷	蛋糕	大树	大旗	大河	大海	大潮	春雨
春风	春潮	春草	船	边角料	翅膀	潮水	潮	风景	长河
波浪	波澜	宝库	包袱	充电房					

2. 标注训练语料

标记的样例如下。

例句：1998030 - 01 - 006 - 020/m 知道/ 所/u 趋/Vg 所/u 赴/v , /w 也/d 就/d 有/v 了/u 决策/v 的/u 坚定性/n 和/e 行动/v 的/u 果敢性/n 。/w 几/m 年/q 来/f , /w 我们/r 数/m 经/v 风雨/n , /w 难题/n 不/d 断/v 。/w 有/v 过/u <% 泡沫/n 经济/n > 的/u 苗头/n 、/w 通货膨胀/l 的/u 现实/n ; /w 遇到/v 过/u 买方/n 、/w 卖方/n 市场/n 的/u 陡然/d 转换/v 和/c 部分/m 企业/n 停产/v 半停产/v 的/u 压力/n ; /w 去年/t 以来/f 更/d 面对/v 东南亚/ns 金融/n 风波/n 。/w 然而/c, /w 无论/e 多么/d 艰难/a , /w < 改革/v 的/u 脚步/n > 却/d 始终/d 都/d 没有/d 停顿/v 过/u , /w 而且/c 总是/d 用/p 改革/v 的/u 办法/n 解决/v 新/a 的/u 难题/n 。/w

标注时要根据“目标域+源域”和“源域+目标域”两种结构确定不同的标注符号。源域在前则标注“<%……>”。例如，<%泡沫/n 经济/n>。源域在后则标注<……>。例如，<改革/v 的/u 脚步/n>。

与单个词语识别相比，n+n 模式识别有如下三点不同。

(1) 把“/n>”作为检索对象。“/n>”标识的隐喻结构中包含多种隐喻表达。

例如，<生活/n 的/u 主线/n>、<精神/n 食粮/n>、<市场/n 的/u 风浪/n>、<改革/n 的/u 春风/n>。

通过检索“/n>”，即可统计出以“主线、食粮、风浪、春风”为核心的隐喻表达在训练语料的分布，而单个词语识别则以“关键词”作为检索对象。如识别以“主线/n”为核心的隐喻表达，则以“主线”为检索对象。

(2) 以“/n>”所在的位置来确定上下文窗口，模型上下文特征不仅包含“/n>”前后的两个词语和词类，也包含“/n>”标注的名词本身。

例句：去年/t 7 月/t 爆发/v 的/u < 东南亚/ns 金融/n 风暴/n > 稍后/d 便/d 波及/v 到/v 了/u 韩国/ns 和/c 日本/ns 。/w

例句：国有/vn 企业/n 本身/r 并/d 不/d 是/v < 社会/n 的/u 包袱/n > , /w 却/d 背着/v 沉重/a 的/u < 社会/n 包袱/n > 。/w

(3) 以“风暴/n>”和“包袱/n>”为核心的前后两个词语和词性分别作为模型的特征，其中也包括“风暴/n>”和“包袱/n>”。模型训练时，只考虑名词源域和目标域组合的情况，即“n+n”结构在文本中的隐喻和非隐喻的分布情况，其他结构类型不予考虑。

例句：在/p 国有/vn 大中型/b 企业/n 的/u 内/f 与/c 外/f 填/v 培/v < 市场经济/n 的/u 土壤/n>。/w

例句：这种/r 有机/b 复合肥/n 不仅/c 肥效/n 高于/v 单/b 元素/n 肥料/n, /w 而且/c 有利于/v 改变/v 目前/t 我国/n 土壤/n 沙化/vn 状况/n。/w

例句：要/v 在/p 那里/r 把/p 土壤/n 里/f 的/u 冰碴/n 提炼/v 成/v 水/n , /w 需要/v 能够/v 在/p 超低温/n 下/f 工作/v 的/u 机器/n , /w 而/c 制造/v 这样/r 的/u 机器/n 是/v 非常/d 困难/a 的/u 。/w

例句：黄山松/n 都/d 长/v 在/p 石头/n 上/f , /w 没有/v 土壤/n , /w

前两例“市场经济/n 的/u 土壤/n”和“我国/n 土壤/n”是 n+n 结构的隐喻和非隐喻用例，可作为识别的考虑对象。后两例“把/p 土壤/n 里/f”和“没有/v 土壤/n”不是 n+n 结构，无法成为 n+n 模式的非隐喻用例，因此被排除。这样排除的好处可以大大提高 n+n 模式在文本中的比例。不足之处可能要忽略其他结构上的隐喻现象。

3. 特征提取

1) 主要特征

无论是最大熵模型还是条件随机场模型，主要特征都是相同的，即上下文窗口的宽度。设置窗口大小分别为 (-1, +1)、(-2, +1)、(-2, +2)。经过测试，(-2, +2) 的结果最好。选取 (-2, +2) 上下文窗口为主要特征。

2) 辅助特征

(1) 左右位置特征：根据喻体位于本体的左侧还是右侧来区分是隐喻还是非隐喻的表达。

- ☐ 在知识的海洋中遨游。“海洋”在“知识”的右侧：隐喻。
- ☐ 我们要利用海洋知识。“海洋在“知识”的左侧：非隐喻。
- ☐ 在科学的海洋中。“海洋”在“科学”的右侧：隐喻。
- ☐ 海洋科学探索。“海洋”在“科学”的左侧：非隐喻。

(2) “的”字特征：n+n 的模式中是否包含“的”。

(3) 标点特征：n+n 的模式中是否包含标点符号。

4. 应用算法

综合全部辅助特征，再次实验。实验前提：已区分左右，不将“标点”作为特征；

将“的”视为特征。最大熵模型窗口+辅助特征的最优结果如表 8.7 所示。

表 8.7 最大熵模型窗口+辅助特征的最优结果

窗口大小	泥沼 (%)	海洋 (%)	港湾 (%)	旋涡 (%)	潮水 (%)
(-1,+1)	P=100.0	P=60.0	P=7.7	P=90.9	P=93.3
	R=83.3	R=58.8	R=33.3	R=100.0	R=93.3
	F=90.9	F=59.4	F=12.5	F=95.2	F=93.3
(-2,+1)	P=100.0	P=73.2	P=40.0	P=90.9	P=93.3
	R=83.3	R=58.8	R=66.7	R=100.0	R=93.3
	F=90.9	F=65.2	F=50.0	F=95.2	F=93.3
(-2,+2) <最优>	P=100.0	P=78.0	P=66.7	P=90.9	P=100.0
	R=83.3	R=62.7	R=66.7	R=100.0	R=100.0
	F=90.9	F=69.6	F=66.7	F=95.2	F=100.0

最大熵方法简单特征的窗口比较实验得到 (-2,+1)、(-2,+2) 窗口要好于 (-1,+1)，通过加入辅助特征，实验效果得到了进一步的提高，(-2,+2) 表现最为突出。因此，结论为：窗口大小取 (-2,+2)，并区分左右，不将“标点”视为特征，将“的”视为特征，得到的识别结果为最优。

8.5 构式语法

转换生成语法的理论基础是“模块观”(Modular View)。模块观认为，语言构成于一系列模块的组合，通过研究语言的形式即可揭示语言的本质，对形式构造的研究可独立于它们的语义和功能；语法研究的对象应是可用规则推导得出的所谓的“核心”部分，而习语、熟语等半规则和不规则的语言结构是边缘现象(Periphery)，可不予理会。

此外，模块观还认为，语言结构相当复杂，不可能依赖一般的认知机制，仅通过少量的输入，用归纳的方式即可学会，人类一定是通过先天赋有的能力才掌握语言的。

在转换生成语法的多年实践中，越来越多难以解释的问题使人们开始对转换生成语法的理论基础产生了怀疑。认知语言学的构式语法就诞生于对乔姆斯基的转换生成语法理论的反思和否定过程之中。认知语言学的主要观点如下。

- ❑ 批判天赋观：语言学习主要是由意义驱动和交互驱动的，不是由头脑中的普遍语法驱动的。

- 批判普遍观: 构式语法遵循着从特殊到一般的规律, 既分析特殊性, 也兼顾普遍性。
- 批判自治观: 语言能力不是天赋的和自治的, 语法知识主要是后天习得的, 是从若干基本用法中抽象出来的, 句法与语义密切相关, 不可分离。
- 批判模块观: 倡导用“象征单位”和“构式”来对语言做出统一的、全面的解释。
- 批判二元论和形式观: 取消一系列的二元对立, 用同一的方法分析语言各层面。

——《构式语法》

从20世纪70年代开始, 一批卓有成就的认知语言学家在对转换生成语法的批判过程中, 涌现了一批颇有成就的认知语言学家, 如 Langacker、Lakoff、Fillmore、Taylor、Goldberg 等, 在他们的带动下, 认知语言学的发展硕果累累, 如图式理论、框架语义学、认知语义学、体验哲学等都为构式语法的提出提供了丰厚的土壤。20世纪90年代, 在认知语言学框架中逐步形成了一种新的语法观——构式观。

对语法的构式观有不同的表述, 但殊途同归, 都一致反对语法的模块观。构式观认为研究语言的形式离不开对意义和功能的审视, 形式和意义是密不可分的结合体; 以一般的语法规则为参照的半规则和不规则结构同样是语言学研究的重要课题, 这些结构是构成人们语言知识库不可或缺的部分, 对它们的研究能够大大拓展人们对语言本质的认识; 人类不是通过先天赋有的能力习得语言的, 赖以掌握语言的输入是十分丰富的, 且习得语言与习得其他知识一样, 都借助于一般的认知能力。

8.5.1 构式的概念

构式语法(Construction Grammar)是20世纪90年代由德国认知语言学家 A.E.Goldberg 提出的。所谓构式(Construction)是指语法中“形式与意义的配对体”。每一种构式都表达了一种句法的语义, 而这种特殊的意义不能从构式的组成成分或已有的其他构式中推断。

这里需要解释一下“形式与意义的配对体”的含义, 其也简称为“形义配对体”。从“象似性”的角度来看, 文字是语言的形, 语义是语言的义。语义与文字的一一对应就是一种“形义配对体”。Langacker 认为两个或两个以上的象征单位所形成的结构才是构式。一个词素就是一个象征单位, 两个或两个以上的词素并置, 经过整合加工后就形成了一个词法或句法上相对复杂的表达, 称为“语法构式”。

例如,“心”是一个象形字,它的本义对应着人的心脏——人身上最重要的器官。用认知学的术语来讲,符号“心”及其语义“心脏”就是一个最简单的象征单位或者一个“形义配对体”。如果“心”表示的意义脱离了“心脏”器官,那么此时的“心”也就失去了原来的意义,如“中心”、“当心”等,由“心”所构造的词汇与“心”本义构造的词汇就不属于同一构式。

这种“形义配对体”的现象不仅存在于造字的环节,也存在于构词法、语义组块、句式等更高级的语言学范畴。构式语法认为语言各层次的表达都是“形义一体”的符号,都可统一于“构式”这一概念之下。从三个平面的角度来看,它从理论上整合了语义和语法,创造了完整的形义合一的语法新结构。这样的语法理论就从句法范畴扩展到了语用范畴。

构式语法的“形义一体”的观念促使我们进一步探索影响句义的组成。不完整的句义组成如图 8.16 所示。

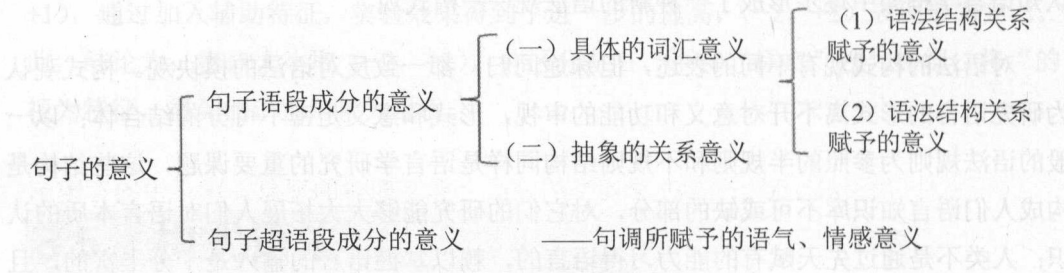


图 8.16 不完整的句义组成

现在看来这种描述并不完整,应修改为如图 8.17 所示的内容。

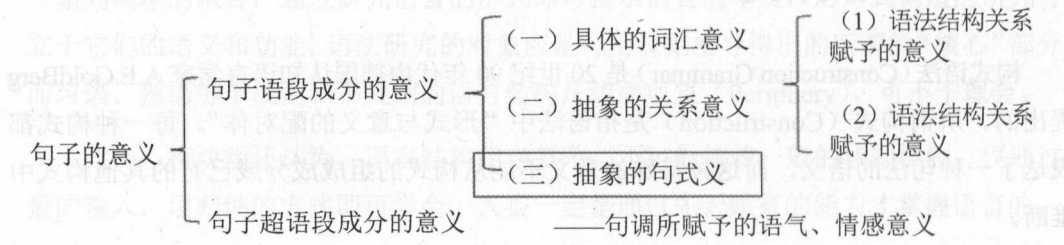


图 8.17 新的句义组成

——陆俭明

注意图 8.17 中的红色边框。边框内的内容就是构式语法所带来的新发现。它说明“构式”映射出的不是概念的语义,而是结构的语义,从本质上来讲它是一种“图式”。这一特殊的语义形式显然来自语用选择的不同,因而构造出特殊的句法形式。这使我们从一

个新的观念和新的视角上看待语言构成的内在意义。

人的认知逻辑结构（象似性、思维的时序关系、空间的层次关系）决定了语言的语义结构，而信息的表达方式决定了语言多种多样的语法规则（社交图式、语用范畴等）。语义结构和语法规则两者可能一致，也可能不一致。而语言表示的多样性就是在这两种不同力量相互作用下的产物。当它们一致的时候，句子的句法分析和语义分析的结果也是一致的，也就是天然的“形义一体”。反映在句子的语义的序关系和句法的序关系的一致性上（受听觉器官的限制，语义的表达为一种序列结构）。当它们不一致的时候，必须通过附加结构加以说明，这种附加结构可能是词汇的形态变化，也可能是通过专门的附加语来说明的。如果使用附加语，那么必然导致句法的序关系和语义的序关系的不一致。作为一种缺乏形态变化的语言，汉语只能选择附加语来重新构架语言的句法结构。

因此，汉语构式研究的一个核心任务就是要解释在汉语原有语法构成的各个层次中，形义不一致的语法成分与语义结构的“配对”问题。从 8.5.2 节开始举例说明一些汉语句中常见的特殊句式。

8.5.2 句法与构式

本节主要介绍构式语法在对抽象句型的分析上形义配对的现象。汉语中的存现句、“把”字句、“被”字句等都是构式，因为它们具有自己独立的意义和功能。近年来，汉语构式句法研究取得了一些重要的成果，这里选取一些比较有代表性的内容呈现给读者。对于下面绝大部分例子都做了句法和语义的标注。有关语义标注的更多细节可参照第 9 章的内容。

□ 中动构式：NP+V-“起来”+AP（AD+JJ）。

- 这本书卖起来很容易。
- 这种地板漆起来很容易。
- 这个程序使用起来很方便。
- 这辆卡车装起来很难。

句法与语义角色的对照如下。

NP	VP	AP（AD+JJ）
Arg0	REL	ArgM- EXT

□ 双及物构式：NP+VP+N1（某人、某处）+N2（某物或某事）。

- 他寄给我一个包裹。

- 他捐了图书馆一套古籍。
- 小张吃了他三个苹果

句法与语义角色的对照如下。

NP	V- “起来”	N1 (某人、某处)	N2 (某物或某事)
Arg0	REL	Arg1	Arg2

□ “把”字句句式：NP1+BA(把)+NP2+VP+OTHER(AP,PP)。

- 看把个大礼拜天搅得!
- 路灯把大马路照得又光又亮。
- 你不能把房子盖到别人家吧。

句法与语义角色的对照如下。

NP1	BA	NP2	VP	OTHER
Arg0	—	Arg1	REL	ArgM-X

□ 致使句式（包含“把”字结构）：NP+SHI+NP+VP+OTHER。

致使方式的语义构成：致使者+致使方式+致使对象+致使结果。

- 土匪使所有房子都烧毁了。

致使者：土匪；致使方式：烧；致使对象：房子；致使结果：毁（动结式结构的分拆）

- 她将这么好的主意白白送给别人。

致使者：她；致使方式：送给；致使对象：主意；致使结果：别人。

- 老师将6看成了9。

致使者：老师；致使方式：看成；致使对象：6；致使结果：9。

句法与语义角色的对照如下。

NP	SHI (使)	NP	VP (双及物、动结式)	OTHER
Arg0	—	Arg1	REL	Arg2或ArgM-EXT

□ 存现句式：NP+VP+NP。

存现句式：存在处所（时间）+存在方式+存在物。

- 花丛里飞来了一只小蜜蜂。

存在处所：花丛；存在方式：飞来；存在物：小蜜蜂。

- 昨天来了一批留学生。

存在时间：昨天；存在方式：来；存在物：留学生。

➤ 最前面是几位海军战士。

存在处所：最前面；存在方式：是；存在物：海军战士。

句法与语义角色的对照如下。

NP	VP	NP
ArgM-LOC(TMP)	REL	Arg0

□ 含有“得”的事件合并构式：NP+V+“得”+NP+VP。

➤ 他拉得风箱吱吱作响。

事件 1：他拉风箱。事件 2：风箱吱吱作响。

句法与语义角色的对照如下。

事件	NP	V	“得”	NP	VP
事件1	Arg0	REL	—	Arg1	—
事件2	—			Arg0	REL

➤ 老百姓都恨得他牙痒痒。

事件 1：老百姓都恨他。事件 2：老百姓牙痒痒。

句法与语义角色的对照如下。

事件	NP	V	“得”	NP	VP
事件1	Arg0	REL	—	Arg1	—
事件1	Arg0	—			REL

8.5.3 构式知识库

随着认知语言学的研究深入，构式语法相关的语料资源积累越来越丰富。本节将介绍一个在建的构式语料库——北大构式知识库，网址为 URL:<http://ccl.pku.edu.cn/ccgd/view.asp>。构式库的构建还在开始阶段，全部知识库共收录汉语中构式总数为 816 条。统计(<http://ccl.pku.edu.cn/ccgd/stat.asp>)页面内给出了构式数据按照各种指标的统计结果。北大构式知识库按照如下 4 个不同的特征选取汉语中的构式。

1) 异常语序

例句：傻瓜一个。

对应的正常语序为“一个傻瓜”。这样的“数+量+名”结构是汉语中偏正结构的常规语序，即修饰语在前、中心语在后。例句对应的“名+数+量”结构是非常规语序，结

构整体也因此有了特殊的意味。

2) 异常组配

例句：(1) 好一个摩登的丫头。

(2) 好你个李云龙。

(3) 那个紧张啊。

在常规组合中，“好”作为形容词修饰名词性成分，一般是单个名词，比如“好人”、“好爸爸”、“好领导”等，不能是复杂的 NP，如“*好我的爸爸”、“*好三个人”等都是不合法的结构。但(1)、(2)中“好”后面都是复杂 NP。类似的，常规组合中，指示代词“那”加量词“个”一般只修饰名词性成分，如“那个司机”、“那个邮箱”等，但(3)中“那个”后面跟的是形容词性成分“紧张”。

3) 成分省略

例句：(1) 好个摩登的丫头。

(2) 一排椅子四个人。

(1) 中量词“个”前面省略了数词(“异常组配”中例句(1)是这里例句(1)的完整形式)。(2)中“一排椅子”和“四个人”之间省略了动词，离开语境，(2)中这两个 NP 之间的关系并不清楚。可能是“一排椅子坐四个人”，也可能是“一排椅子站四个人”、“一排椅子躺四个人”等。

4) 同型复现

例句：(1) 他到了救灾现场，就泥一把水一把地跟大伙一块儿干了起来。

(2) 她的病情一天比一天严重了起来。

(1) 和(2)中都有重复的“数+量”形式。上述例句中的“一排椅子”和“四个人”也是一种“同型”，只不过不是完全“同形”，而是短语功能类层次上的“同型”：都是“数+量+名”构造。

——《现代汉语构式知识库的构建与应用》

构式语料库截图如图 8.18 所示。

浏览数据库

序号	构式形式	构式类型	构式特征	变项数量	常项数量	实例	更多操作
1	a+m+q	短语型	错配	3	0	长三丈 宽三尺 粗一指	详细
2	a+巴巴	半凝固型	错配	1	1	可怜巴巴 皱巴巴 干巴巴 瘦巴巴	详细
3	a+爆	短语型	错配	1	1	爽爆 潮爆 HIGH爆	详细
4	a+不+到+哪里+去	短语型	错配	1	4	好不到哪里去 富裕不到哪里去 高不到哪里去	详细
5	a+不+死+ <i>r</i>	短语型	错配	2	2	瘦不死她 烫不死你 饿不死我	详细
6	a+不堪	半凝固型	异序 错配	1	1	痛苦不堪 困乏不堪 饥饿不堪	详细
7	a+到+爆	短语型	错配	1	2	累到爆 爽到爆 傻到爆	详细
8	a+到+不+能+再+a	短语型	复现	2	4	好到不能再好 低到不能再低 细到不能再细	详细
9	a+得+vp	短语型		2	1	贵得要命 疼得要死 少得可怜人	详细
10	a+得+很	短语型		1	2	好得很 冷得很 清楚得很	详细

图 8.18 构式语料库截图

图 8.18 中的部分内容说明如下。

(1) 构式类型。

□ 凝固型：构式中不含可变量的数量，只有常项的数量。

例句：半+毛+钱+关系、抽屉+协议、打+光棍、打+主意、给+跪+了。

□ 半凝固型：含有变量的构式，在“构式形式”一栏中小写字母为变量的词性，字母的位置为变量位置。

➤ a+巴巴：可怜巴巴|皱巴巴|干巴巴|瘦巴巴。

➤ a+不堪：痛苦不堪|困乏不堪|饥饿不堪。

➤ n+百+出：漏洞百出|错误百出|洋相百出。

➤ n+帝：牢骚帝|表情帝|数学帝。

□ 复句型：含有变量的复句型的构式。

➤ a1+的+a1, a2+的+a2：大的大，小的小|漂亮的漂亮，优雅的优雅|高的高，矮的矮。

➤ n+并不+vp1, 也+不+vp2：他并不想升官，也不想发财|他并不是第一个，也不会是最后一个|二牛并不哭，也不走。

➤ n1+不+是+n1, n2+不+是+n2：鼻子不是鼻子，脸不是脸|喜剧不是喜剧，闹剧不是闹剧。

□ 短语型：含有变项的短语型的构式。

- a+m+q: 长三丈|宽三尺|粗一指。
- a+爆(短语型): 爽爆|潮爆|HIGH 爆。
- a+不+到+哪里+去: 好不到哪里去|富裕不到哪里去|高不到哪里去。

(2) 构式特征。

- 异序：语序颠倒。例句：白了头发|红了脸|麻了一只脚。
- 错配：错位的搭配。例句：瘦不死她|爽爆|拿不定。
- 省略：构式中有省略的成分。例句：三个人一锅饭|两排椅子六个人。
- 复现：构式中出现重复的成分。例句：青一块，紫一块|看也不看|想也不想|专而又专。
- 冗余：构式中在语义上有冗余成分。例句：半生不熟|大吃特吃|半死不活。
- 论元异常：构式中谓词的论元与常态不同。例句：我偷他两块钱|我吃他两个苹果|拿小王没办法。
- 空特征：为给定具体特征的构式。例句：好得很|犬牙交错|拂袖而去。

与隐喻一样，构式也揭示出了语言的复杂性。目前的主要语种都难以用一种简单的基于规则的句法体系涵盖进来。一方面它需要大规模的语料支持，从语料中发掘出尽可能多的模式；另一方面需要更深刻地理解“象似性”原理，并以此为基点定义出与之相符的句法结构和智能算法。

8.6 结语

本章是本书关于语言理论的核心章节，阐述了认知语言学对传统语言学的新发展和新突破，并通过大量的语言现象来解释认知的原理。本章在整体上分为如下两大部分。

第一部分也就是8.1节回顾了语义研究的历史。首先简单阐述了语义三角论的思想，这一思想是语义诞生的标志性里程碑。接着简述了各个时期语义理论的发展，如语义场理论及基于逻辑的语义学。

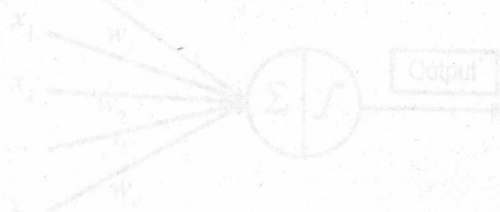
从第二部分开始详细介绍了认知语言学的内容。首先介绍了认知的基础——“象似性”原理。然后以此为出发点，逐步展开到意象图式理论。重点讲解了主观性与焦点、范畴化、主体与背景、意象图式、社交中的图示，以及完形这些概念。并都给出了语言上的例证。

之后,介绍了隐喻与转喻理论,这是“象似性”原理的直接体现。除此之外,还给出了隐喻计算的系统架构及其实现。

最后,讲解了构式语法,并通过句法与构式的比较加深了读者对构式语法的认识,以及引入了北京大学的构式知识库,并做了详细的介绍。

9.1.1 神经网络框架

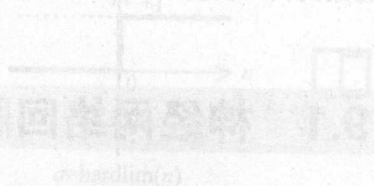
图9.1所示为一个最简单的感知器神经网络,也称为神经元。



神经网络是由大量的人工神经元或节点按某种方式连接而形成的。它是一种模仿人脑神经网络而设计的计算模型。神经网络的基本单元是神经元,它由输入、隐藏和输出层组成。每个神经元接收来自前一层神经元的信号,并根据一定的权重进行加权求和,然后将结果传递给下一层神经元。这种结构使得神经网络能够处理复杂的非线性问题。

神经网络的学习过程通常分为训练和测试两个阶段。在训练阶段,神经网络会根据给定的训练数据调整其内部权重,以最小化损失函数。常用的训练算法包括反向传播和梯度下降法。在测试阶段,神经网络会根据测试数据输出预测结果,并计算其准确率。神经网络的广泛应用包括图像识别、语音识别、自然语言处理等领域。

如图9.2所示,最早使用的激活函数是硬限制(hardlim)函数。硬限制的含义是,对于一个二元分类问题,可以设想成利用一个超平面将输入空间划分为两个区域。在超平面一侧的所有点都被分类为“1”,另一侧则被分类为“0”。这种简单的二元分类模型是神经网络的基础。



第 9 章

NLP 中的深度学习

本章将给出 NLP 的第二个算法体系：基于人工神经网络的深度学习理论。人工神经网络思想来源于大脑机制的探索，即对大脑思维能力的研究和模仿。神经网络理论与相关技术就是为了实现思维的认识机能而发展出来的，长久以来，它都是这门学科的基本任务。

自 2006 年以来，Geoffrey Hinton 在深度学习上获得了重大的突破，他与他的深度学习理论将人工智能带入了一个新的时代：认知计算。认知计算的目标不再是寻求现实问题中的最优解或在给定的数据结构上进一步提高搜索性能，而是把算法领域扩展到了探索大脑的深度机制——认知机制方面。它的目标是让计算机系统能够像人类的大脑一样学习、思考，并做出正确的决策。从此，算法研究与认知神经科学、心理学、脑科学等自然科学紧密地联系起来。

这些新生的算法在自然语言处理领域同样获得了巨大的成功。本章从人工神经网络开始，逐层深入地介绍目前在 NLP 中比较流行的深度学习算法思想，并尽可能地提供一些案例代码以便于读者进一步地学习和实践。

9.1 神经网络回顾

本节将从感知器网络开始介绍神经元的结构和最简单的神经网络：感知器网络，然

后通过一个实例详细讲解梯度下降法实现神经网络的优化过程。

接下来,简要介绍 BP 神经网络的架构,以及反向传播的推导过程。为后面的深度学习中的 RNN 网络奠定基础。

9.1.1 神经网络框架

图 9.1 所示为一个最简单的感知器神经网络,也称为神经元。

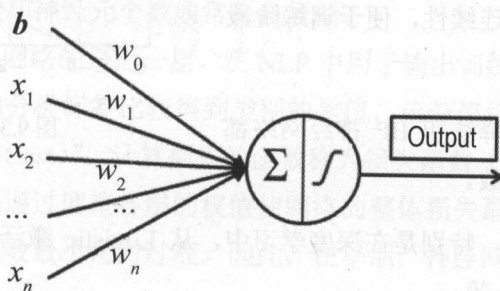


图 9.1 神经元

后面所述的各种神经网络都是由这一最基本的结构构成的。神经网络中任意一个神经元都可以表示为上述的形式。一个神经元由如下两部分构成。

(1) 带有权重的自由变量。该自由变量的取值来自前一层所有变量与权重的乘积,然后再求和,在数学上表示为如下形式。

$$x^j = \sum_{i=1}^n x_i^{j-1} w_i^{j-1}$$

其中, x 为自由变量, x^j 为当前层, 这里 $x_i^{j-1} w_i^{j-1}$ 为前一层所有变量与权重的乘积, n 为神经元个数。在实践中,神经网络的输入层由训练样本给定,隐含层和输出层的 x 取值由前一层计算得到。

(2) 一个决策分类的激活函数。

如图 9.2 所示,最早使用的激活函数是硬限幅(hardlim)函数。硬限幅的含义是,对于一个二元分类问题,可以设想成利用一个超平面划分两个高维凸集空间的情况:在超平面一侧的所有点集都被分类为“1”,另一侧则被分类为“0”。

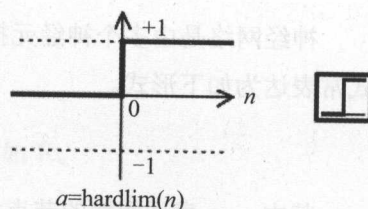


图 9.2 hardlim 硬限幅函数

在实践中，由于硬限幅函数在训练时难以确定决策边界，而且它是一个不连续的函数，无法对复杂网络数据模型求导，后期人们对该函数做了各种修正。

如图 9.3 所示，其中最成功、应用最广泛的一种激活函数是 Logistic 函数。该函数的优势是，首先它放大了决策边界，类别的决策过程使用一条平滑的“S”形曲线来代替间断的硬限幅函数。其次，Logistic 曲线是连续的，在数学上保证了整个函数的连续性，便于训练阶段的求导。

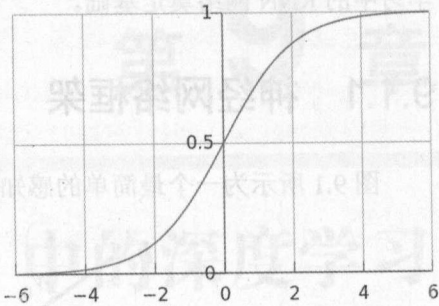


图 9.3 Logistic 函数

后面所讲的梯度下降法和 BP 神经网络都用这个函数作为激活函数。

神经网络发展至今，特别是在深度学习中，从 Logistic 激活函数衍生出多种激活函数，包括 tanh、softmax 等。

神经网络，也称为人工神经网络，是一个分层结构，在两个层次之间的神经元两两互联所构成的一个网络架构，如图 9.4 所示。

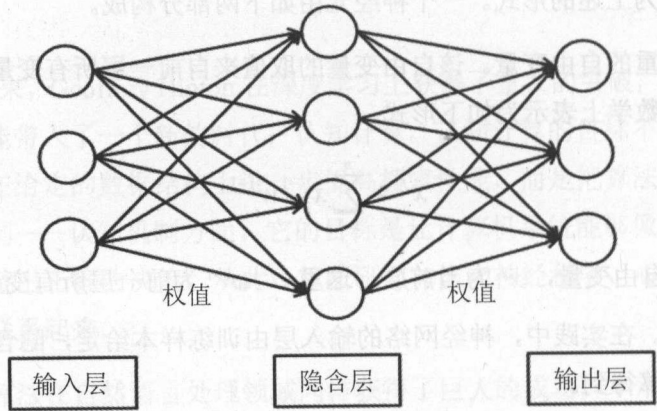


图 9.4 神经网络示意图

神经网络是由多个神经元按层次互联构成的一种计算架构。网络的各层次的数学形式常表达为如下形式。

$$\text{net} = \mathbf{w}^T \mathbf{x} + \mathbf{b}$$

其中， \mathbf{w} 表示网络的节点权重向量； \mathbf{x} 为各层节点的数据（输入层为样本维度）； \mathbf{b} 为偏执向量，默认初始值一般为一个全 1 的列向量。

一般神经网络的层次分为如下三大类。

- 输入层：神经网络的第一层，网络的神经元的个数，称为输入层的维度。该维度来自训练样本的维度（列数），也就是特征数。权重在网络训练开始分配一个初始值，网络模型不同，对初始值的要求也不同。感知器网络一般分配的值 1，BP 网络的分配值为一个 0~1 之间的随机数。
- 隐含层：可以是零层，也可以是一层或多层。不同网络模型的隐含层层数都不同，即便相同的网络也会有不同的隐含层设计。隐含层的神经元个数也没有一定之规。层数和神经元个数通常来自经验。
- 输出层：神经网络的最后一层，在 NLP 中用于输出训练后的预测标签。该预测结果与实际的分类标签比较得到类别的差值，该差值称为“损失（Loss）”，或称为“成本（Cost）”。计算损失的函数称为损失函数（Loss Function），网络的学习过程就是通过调整各层的权值使网络的整体损失最小的过程，换句话说，就是使损失函数最小化的过程。因此，在早期，神经网络借鉴了很多最优化问题的解决方法。

这样，整个神经网络构成了一个算法架构。该架构的执行分为如下两个阶段。

- 训练阶段。是指网络输入样本数据作为初始数据，通过激活函数与网络连接，迭代求得最小化的损失。这时最终学习出权重向量 w ，作为分类器的参数。数学上称这个过程为参数估计的过程。在 NLP 中如果用于序列标注，则可称为一个标注器。
- 分类阶段。拿这个学习出的分类器对实际的数据进行分类或标注，称为分类阶段。

9.1.2 梯度下降法推导

在作为神经网络的算法框架之前，梯度下降法一般作为最优化问题的一种求解方法，是求解无约束多元函数极值的最早的数值方法。下面讲解梯度下降法的基本原理。神经网络的基本公式如下。

$$\text{net} = w^T x + b$$

把公式展开写成标量的形式，让 $b=w_0$ ，公式变形为如下。

$$\text{net}(x) = w_0 + w_1 x_1 + w_2 x_2 + \dots$$

其中, $X=[1, x_1, x_2, \dots, x_{n-1}]$ 为行数为 n 的样本向量, $W=[w_0, w_1, w_2, \dots, w_{n-1}]$ 为列数为 n 的列向量。训练集的观测样本总数为 m 个, 观测值分别为 y_1, y_2, \dots, y_m 。

Logistic 在纵坐标的取值范围是 $(0, 1)$, 横坐标的取值范围是从负无穷到正无穷, 其函数表达式如下。

$$\text{logistic} = \frac{1}{1 + e^{-w^T x}} \quad \text{或} \quad \text{logistic} = \frac{1}{1 + \exp(-\text{net})}$$

为了方便, 把训练集矩阵看作样本总体, 对于总体的每个样本而言, 它们彼此之间都是相互独立的, 其输出标签为 $y=\{0, 1\}$, 那么可以写为如下内容。

如果 $y=1$, 概率就是 p , $p=P(Y=1|x)$, 则令:

$$P\{Y=1|x\} = p(x) = \frac{1}{1 + e^{-w^T x}}$$

如果 $y=0$, 概率就是 $1-p$, $1-p=P(Y=0|x)$, 那么:

$$P\{Y=0|x\} = 1 - p(x) = 1 - \frac{1}{1 + e^{-w^T x}} = \frac{1}{1 + e^{w^T x}}$$

所以, 事件的发生概率与不发生的概率之比为

$$\frac{P\{Y=1|x\}}{P\{Y=0|x\}} = \frac{p(x)}{1-p(x)} = e^{w^T x}$$

两边取对数:

$$\text{logit}(x) = \ln \frac{P\{Y=1|x\}}{P\{Y=0|x\}} = \ln \frac{p(x)}{1-p(x)} = w^T x$$

这样可以得到一个观测值的概率为 $P(y_i) = p_i^{y_i} (1-p_i)^{1-y_i}$ 。因为各个观测样本之间相互独立, 所以它们的联合分布为各边缘分布的乘积, 得到似然函数为

$$l = \prod_{i=1}^n (P\{y_i=1|x_i\})^{y_i} (1-P\{y_i=1|x_i\})^{1-y_i}$$

我们的目标是求出使这一似然函数的值最大的参数估计, 最大似然估计就是求出参数 $w_0, w_1, w_2, \dots, w_{n-1}$, 使得 $L(w) = \ln(l)$ 。上式变换为如下。

$$L(w) = \ln(l(w)) = \sum_{i=0}^{n-1} y_i \ln(p(x_i)) + (1-y_i) \ln(1-p(x_i))$$

继续对这 n 个 w_i 分别求偏导, 得到 n 个方程, 关于 w_k 的推导过程如下。

$$\begin{aligned}
\frac{\partial L(w)}{\partial w_k} &= (y \frac{1}{p(x)} - (1-y) \frac{1}{1-p(x)}) \frac{\partial p(x)}{\partial w_k} \\
&= (y \frac{1}{\text{logistic}(w^T x)} - (1-y) \frac{1}{1-\text{logistic}(w^T x)}) \frac{\partial \text{logistic}(w^T x)}{\partial w_k} \\
&= (y \frac{1}{\text{logistic}(w^T x)} - (1-y) \frac{1}{1-\text{logistic}(w^T x)}) \text{logistic}(w^T x) (1 - \text{logistic}(w^T x)) \frac{\partial w^T x}{\partial w_k} \\
&= (y(1 - \text{logistic}(w^T x)) - (1-y)\text{logistic}(w^T x)) x_k \\
&= (y - p(x)) x_k
\end{aligned}$$

使用递推公式得到:

$$w^i = w^{i-1} - \alpha \Delta L(w) = w^{i-1} - \alpha \frac{\partial L(w)}{\partial w} = w^{i-1} + \alpha [y^{i-1} - p(x^{i-1})] x_j^{i-1}$$

其中, α 就是下降的速度, 一般是一个小的数值, 可以从 0.001 开始尝试, 其值越大表示下降越快、收敛越快。

迭代终止的条件如下。

$$\|w^i - w^{i-1}\| < \varepsilon$$

9.1.3 梯度下降法的实现

本节给出梯度下降法的代码实现。使用的测试数据集可从 <http://www.threedweb.cn/thread-1603-1-1.html> 下载。

(1) 辅助函数: 位于 `common_libs` 中。

① 加载数据文件并转矩阵。

```
# path: 数据文件路径
# delimiter: 文件分隔符
def file2matrix(path, delimiter):
    recordlist = []
    fp = open(path, "rb") # 读取文件内容
    content = fp.read()
    fp.close()
    rowlist = content.splitlines() # 按行转换为一维表
# 逐行遍历, 结果按分隔符分割为行向量
    recordlist = [map(eval, row.split(delimiter)) for row in rowlist if row.strip()]
    return mat(recordlist) # 返回转换后的矩阵形式
```

② 绘制分类点。

```
# 绘制分类点
def drawScatterbyLabel(plt, Input):
    m, n = shape(Input)
    target = Input[:, -1]
    for i in xrange(m):
        if target[i] == 0:
            plt.scatter(Input[i, 0], Input[i, 1], c='blue', marker='o')
        else:
            plt.scatter(Input[i, 0], Input[i, 1], c='red', marker='s')
```

③ 构建 $B+X$ 矩阵，默认 B 为全 1 的列向量。

```
def buildMat(dataSet):
    m, n = shape(dataSet)
    dataMat = zeros((m, n))
    dataMat[:, 0] = 1
    dataMat[:, 1:] = dataSet[:, :-1]
    return dataMat
```

④ Logistic 函数如下。

```
# Logistic 函数
def logistic(wTx):
    return 1.0 / (1.0 + exp(-wTx))
```

(2) 主程序。

```
# -*- coding: utf-8 -*-
import os, sys
import numpy as np
from numpy import *
from common_libs import *
import matplotlib.pyplot as plt

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
# 1. 导入数据
Input = file2matrix("testSet.txt", "\t")
labels = Input[:, -1] # 获取分类标签列表
[m, n] = shape(Input)
# 2. 构建 b+x 系数矩阵: b 默认为 1
dataMat = buildMat(Input)
# print dataMat
```

```

# 3. 定义步长和迭代次数
alpha = 0.001 # 步长
steps = 500    # 迭代次数
weights = ones((n,1)) # 初始化权重向量
errorlist = []

# 4. 主程序
for k in xrange(steps):
    net = dataMat*mat(weights) # 待估计网络
    output = logistic(net) # logistic 函数
    loss = output-labels
    error = 0.5*sum(multiply(loss,loss)) # loss function
    errorlist.append(error)
    grad = dataMat.T*loss # 梯度
    weights = weights - alpha*grad # 迭代

print weights # 输出训练后的权重

# 5. 绘制训练后超平面
drawScatterbyLabel(plt,Input) # 按分类绘制散点图
X = np.linspace(-5,5,100)
Y=- (double(weights[0])+X*(double(weights[1])))/double(weights[2])
plt.plot(X,Y)
plt.show()

```

分类超平面如图 9.5 所示。

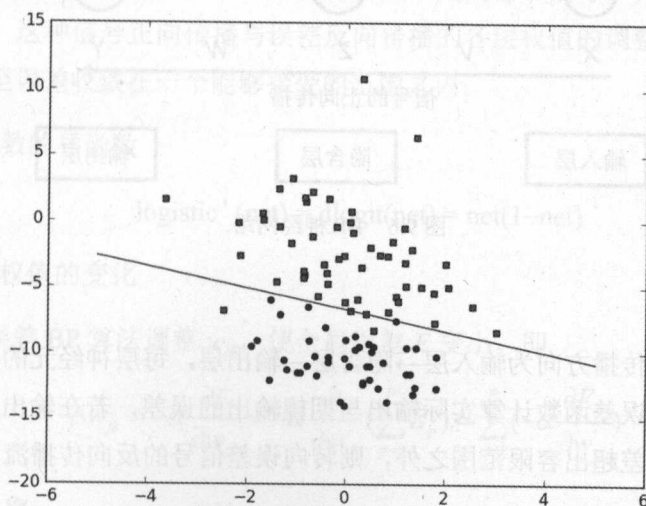


图 9.5 分类超平面

输出的分类超平面向量如下。

```
[[ 4.17881308] [ 0.50489874] [ 0.61980264]]
```


9.1.4 BP 神经网络介绍和推导

BP 神经网络是由 Geoffrey Hinton 提出的一种神经网络算法，它由一个多层感知器结构及 Back-Propagation 训练算法构成，在 20 世纪 80~90 年代鼎盛一时。

BP 算法的基本思想是，学习过程由信号的正向传播与误差的反向传播两个过程组成。正向传播时，输入样本从输入层传入，经各隐层逐层处理后，传向输出层。若输出层的实际输出与期望输出存在差异，则转入误差的反向传播阶段。误差反传是将输出误差以某种形式通过隐含层向输入层逐层反传，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，此误差信号即作为修正各单元权值的依据。这种信号正向传播与误差反向传播的各层权值调整过程，是周而复始地进行的。权值不断调整的过程，是由 BP 算法由数据流的前向计算（正向传播）和误差信号的反向传播两个过程共同构成的。BP 神经网络如图 9.6 所示。

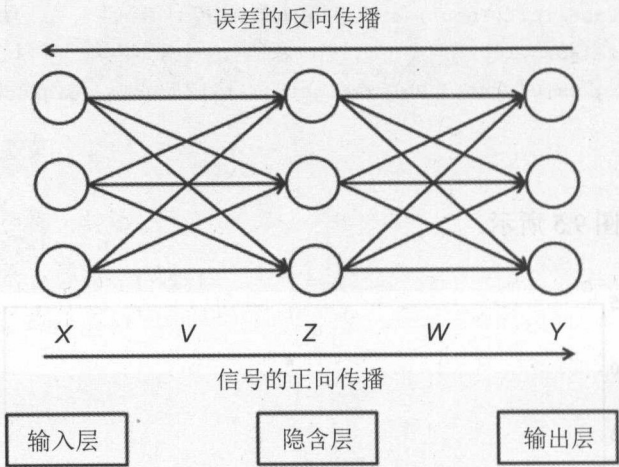


图 9.6 BP 神经网络

1. 正向传播

正向传播时，传播方向为输入层→隐含层→输出层，每层神经元的状态只影响下一层神经元。再通过误差函数计算实际输出与期望输出的误差，若在输出层得不到期望输出或与实际类别的差超出容限范围之外，则转向误差信号的反向传播流程。

1) 正向传播的过程

假设 BP 神经网络的输入层有 n 个节点，隐含层有 q 个节点，输出层有 m 个节点，输入层与隐含层之间的权值为 v_{ih} ，隐含层与输出层之间的权值为 w_{ho} 。隐含层和输出层的激活函数都为 Logistic 函数，则隐含层节点的输出如下。

$$z_k = \text{logistic}\left(\sum_{i=0}^{n-1} v_{ki} x_i\right) \quad k=1,2,\dots,q \quad (1)$$

输出层节点的输出为:

$$y_j = \text{logistic}\left(\sum_{k=0}^{q-1} w_{jk} z_k\right) \quad j=1,2,\dots,m \quad (2)$$

至此, BP 神经网络就完成了 n 维空间向量对 m 维空间的近似映射。

2) 定义误差函数

输入 P 个学习样本, 用 $X=[x^1, x^2, \dots, x^p]$ 来表示。第 p 个样本输入到网络后得到输出 y_j^p ($j=1,2,\dots,m$)。采用平方型误差函数, 于是得到第 p 个样本的误差为 $E_p = \frac{1}{2} \sum_{j=1}^m (t_j^p - y_j^p)^2$

上式中, t_j^p 为预测输出。

$$\text{对于 } P \text{ 个样本, 全局误差为 } E = \frac{1}{2} \sum_{p=1}^P \sum_{j=1}^m (t_j^p - y_j^p)^2 = \sum_{p=1}^P E_p \quad (3)$$

2. 反向传播

反向传播是将误差通过输出层→隐含层→输入层逐层反传, 并通过激活函数的导函数将误差分摊给各层的所有单元, 从而获得各层单元的修正信号, 并以信号作为依据修正各单元权值。这种信号正向传播与误差反向传播的各层权值的调整过程, 是周而复始地进行的, 直至误差收敛在一个能够接受的范围之内。

1) 激活函数的导函数

$$\text{logistic}'(\text{net}) = \text{dlogit}(\text{net}) = \text{net}(1-\text{net}) \quad (4)$$

2) 输出层权值的变化

采用累计误差 BP 算法调整 w_{jk} , 使全局误差 E 变小, 即

$$\Delta w_{jk} = -\alpha \frac{\partial E}{\partial w_{jk}} = -\alpha \frac{\partial}{\partial w_{jk}} \left(\sum_{p=1}^P E_p \right) = \sum_{p=1}^P \left(-\alpha \frac{\partial E_p}{\partial w_{jk}} \right)$$

式中, α 为学习率。

$$\text{定义误差信号为 } \delta_{yj} = -\frac{\partial E_p}{\partial s_j} = -\frac{\partial E_p}{\partial y_j} \cdot \frac{\partial y_j}{\partial s_j} \quad (5)$$

其中, 第一项: $\frac{\partial E_p}{\partial y_j} = \frac{\partial}{\partial y_j} [\frac{1}{2} \sum_{j=1}^m (t_j^p - y_j^p)^2] = -\sum_{j=1}^m (t_j^p - y_j^p)$

第二项: $\frac{\partial y_j}{\partial S_j} = \text{logistic}'(S_j) = \text{dlogit}(S_j)$ 是输出层传递函数的偏微分。

根据链定理, 第一项与第二项相乘得到: $\delta_{yj} = -\sum_{j=1}^m (t_j^p - y_j^p) \text{dlogit}(S_j)$ (6)

Δw_{jk} 中的 $\frac{\partial E_p}{\partial w_{jk}}$ 推导可得: $\frac{\partial E_p}{\partial w_{jk}} = \delta_{yj} \cdot z_k = -\sum_{j=1}^m (t_j^p - y_j^p) \text{dlogit}(S_j) z_k$

于是, 输出层各神经元的权值调整公式如下。

$$\Delta w_{jk} = \sum_{p=1}^P \sum_{j=1}^m \alpha (t_j^p - y_j^p) \text{dlogit}(S_j) z_k \quad (7)$$

3) 隐含层权值的变化

$$\Delta v_{ki} = -\alpha \frac{\partial E}{\partial v_{ki}} = -\alpha \frac{\partial \sum_{p=1}^P E_p}{\partial v_{ki}} = \sum_{p=1}^P (-\alpha \frac{\partial E_p}{\partial v_{ki}})$$

定义误差信号为 $\delta_{zk} = -\frac{\partial E_p}{\partial S_k} = -\frac{\partial E_p}{\partial z_k} \cdot \frac{\partial z_k}{\partial S_k}$ (8)

其中, 第一项: $\frac{\partial E_p}{\partial z_k} = \frac{\partial}{\partial z_k} [\frac{1}{2} \sum_{j=1}^m (t_j^p - y_j^p)^2] = -\sum_{j=1}^m (t_j^p - y_j^p) \frac{\partial y_j}{\partial z_k}$

依据链定理有: $\frac{\partial y_j}{\partial z_k} = \frac{\partial y_j}{\partial S_j} \cdot \frac{\partial S_j}{\partial z_k} = \text{dlogit}(S_j) w_{jk}$

第二项: $\frac{\partial z_k}{\partial S_k} = \text{dlogit}(S_k)$ 是隐含层传递函数的偏微分。

第一项与第二项相乘得到: $\delta_{zk} = \sum_{j=1}^m (t_j^p - y_j^p) \text{dlogit}(S_j) w_{jk} \text{dlogit}(S_k)$ (9)

由链定理得: $\frac{\partial E_p}{\partial v_{ki}} = \frac{\partial E_p}{\partial S_k} \cdot \frac{\partial S_k}{\partial v_{ki}} = -\delta_{zk} x_i = -\sum_{j=1}^m (t_j^p - y_j^p) \text{dlogit}(S_j) w_{jk} \text{dlogit}(S_k) x_i$

从而得到隐含层各神经元的权值调整公式为如下。

$$\Delta v_{ki} = -\sum_{p=1}^P \sum_{j=1}^m (t_j^p - y_j^p) \text{dlogit}(S_j) w_{jk} \text{dlogit}(S_k) x_i \quad (10)$$

由于篇幅限制,这里就不给出 BP 网络的案例代码,对 BP 网络实现感兴趣的朋友可以参考笔者所著的另一本书——《机器学习算法原理与编程实践》中的实现。

3. 应用不足

BP 网络自诞生至今,大大小小的改进版本不计其数,一度是应用最广泛的神经网络算法。随着大量的应用也暴露出了如下问题。

首先,由于学习速率 α 是固定的,因此网络的收敛速度慢,需要较长的训练时间。对于一些复杂问题, BP 算法需要的训练时间可能非常长,这主要是由于学习速率太小造成的,可采用变化的学习速率或自适应的学习速率加以改进。

其次, BP 算法可以使权值收敛到某个值,但并不保证其为误差平面的全局最小值。这是因为采用梯度下降法可能产生一个局部最小值,网络容易陷入局部最优。

再次,网络隐含层的层数和单元数的选择尚无理论上的指导,一般是根据经验或者通过反复实验确定的。因此,网络往往存在很大的冗余性,在一定程度上也增加了网络学习的负担。

最后,网络的学习和记忆具有不稳定性。也就是说,如果增加了学习样本,训练好的网络就需要从头开始训练,对于以前的权值和阈值是没有记忆的,但是可以将预测、分类或聚类做得比较好的权值保存。

9.2 Word2Vec 简介

Word2Vec 是 Google 公司于 2013 年发布的一个开源词向量工具包。该项目的算法理论参考了 Bengio 在 2003 年设计的神经网络语言模型。由于此神经网络模型使用了两次非线性变换,网络参数很多,训练缓慢,因此不适合大语料。Mikolov 团队对其做了简化,实现了 Word2Vec 词向量模型。它简单、高效,特别适合从大规模、超大规模的语料中获取高精度的词向量表示。因此,项目一经发布就引起了业界的广泛重视,并在多种 NLP 任务中获得了良好的效果,成为 NLP 在语义相似度计算中的重大突破。

Word2Vec 及同类的词向量模型都基于如下假设:衡量两个词在语义上的相似性,决定于其邻居词分布是否类似。显然这是源于认知语言学中的“距离象似性”原理:词汇与其上下文构成了一个“象”。当从语料中训练出相同或相近的两个“象”时,无论这两个“象”的中心词汇在字面上是否一致,它们在语义上都是相似的。

自从 Word2Vec 框架发布之后,无论是在国外还是在国内,该框架都引起了巨大的反响。由于 Tomas Mikolov 在相关的论文中并没有谈及太多的算法细节,因此对许多 NLP 的研究人员来说,对该算法的研究一度成为重要的课题。经过两三年的研究,到目前为止,根据发布出来的研究成果,对相关理论的研究已经非常充分,在网络上可以很容易地找到。同时,人们在研究的过程中还对源码做了详尽的注释,感兴趣的读者可以从 <http://www.threedweb.cn/thread-1598-1-1.html> 下载带注释的版本,便于学习。

为了使内容具有完整性,本章仅从神经网络的角度对 Word2Vec 进行简单介绍。

9.2.1 词向量及其表达

数学上,词向量可以表示为对词典 D 中的任意词 w , 指定一个固定长度的实值向量 $\mathbf{v}(w) \in R^m$ 。 $\mathbf{v}(w)$ 就称为 w 的词向量, m 为词向量的长度。

使用向量来表示词最早是 Hinton 在 1986 年的论文 *Learning Distributed Representations of Concepts* 中提出的。几年以后,传统的神经网络算法逐渐走向了低潮期,这种思想渐渐被人们所忘却,到 2000 年之后又开始引起了人们的重视。

第 4 章简要介绍过 One-Hot 词向量的概念,这里所说的词向量的分布式表达 (Distributed Representation) 与第 4 章的概念不同,它克服了 One-Hot 词向量的两大缺陷。首先是人们所说的“词汇鸿沟”现象,One-Hot 的基本假设是,词与词之间的语义或语法关系是相互独立的,仅从这两个向量中看不出两个词是否有关系,这种独立性不适合词汇语义的比较运算;其次是人们所说的“维度灾难”,随着词典规模的增大,句子构成的词袋模型的维度变得越来越大,非零值的分布却越来越稀疏。这种维度的激增会对计算提出更高的要求。

词向量的 Distributed Representation 的优势在于将语言中的每一个词映射成一个固定长度的短向量,形如 $[0.054\ 656\ -0.220\ 857\ 0.120\ 756\ -0.170\ 938\ 0.215\ 805\ \dots]$, 用来克服 One-Hot 表达的上述缺陷。在 Word2Vec 中,这个短向量的维度是自定义的,默认是 100 维,根据训练语料的规模可以扩充或收缩(规模大一些的 100 维,小一些的 50 维)。

所有这些向量构成一个词向量空间,而每一个向量则视为该空间中的一个点,这样即可引入“距离”的概念,通过计算词之间的距离(如余弦相似度、欧氏距离等)来判断它们之间的语义相似度。

笔者参考一些相关的文献,给出如下例子。

图 9.7 所示为著名的跨语言同义词共现的一个案例,很多文献都在用,本书也不能

免俗。主要因为这个案例能够透彻地说明问题。该案例源于 Tomas Mikolov 团队开发的一项语义映射技术，它是一种特殊的机器翻译系统。它通过为两种语言构建不同的语言空间，并在两个空间上建立映射关系，只要实现一个向量空间向另一向量空间的映射和转换，语言翻译即可实现。该技术效果非常不错，英语对西班牙语的机器翻译准确率可达 90%。

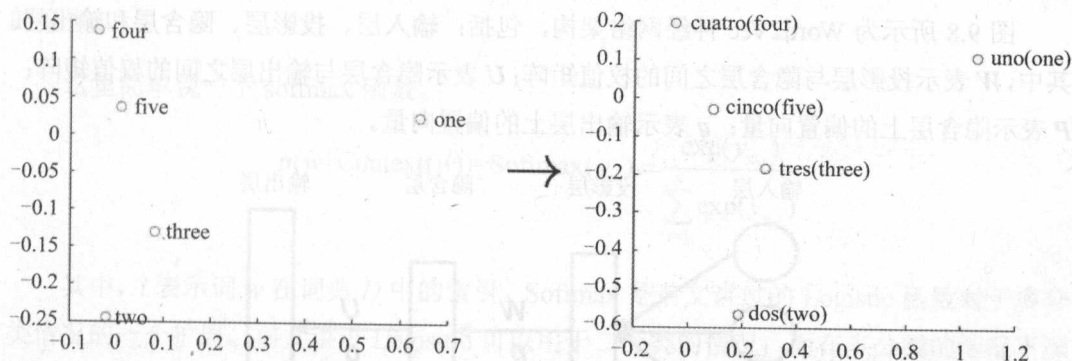


图 9.7 著名的跨语言同义词共现的一个案例

例如，将英语和西班牙语两种语言通过 Word2Vec 训练产生词的分布表示向量。不同语言的词向量分别得到它们对应的词向量空间 English（图 9.7 左图）和 Spanish（图 9.7 右图）。从英语中取出 5 个词 one、two、three、four、five，为方便作图，利用主成分分析（PCA）降维，得到相应的二维向量 P_{one} 、 P_{two} 、 P_{three} 、 P_{four} 、 P_{five} 。在二维平面上将这 5 个点描出来，如图 9.7 左图所示。

类似的，在西班牙语中取出 uno、dos、tres、cuatro、cinco 5 个点（与 one、two、three、four、five 对应的）。设其在图 9.7 右图中对应的词向量，用 PCA 降维后的二维向量分别为 P_{uno} 、 P_{dos} 、 P_{tres} 、 P_{cuatro} 、 P_{cinco} 。将它们在二维平面上描出来（可能还需进行适当旋转），如图 9.7 右图所示，观察左、右两幅图，容易发现：5 个词在两个向量空间中的相对位置差不多。这说明两种不同语言对应向量空间的结构之间具有相似性。从而进一步说明了在词向量空间中利用距离刻画词之间相似性的合理性。

词向量的评价大体上可以分成两种方式：一种是把词向量融入现有系统中，提升现有系统的性能；另一种是直接从语言学的角度对词向量进行分析，如相似度、语义偏移等。

训练一份好的词向量对于之前我们讲过的 NLP 系统很有价值，特别对于深度学习而言。基于词向量的 LSTM 架构在中文分词、词性标注、语义组块、命名实体识别方面都取得了良好的效果，在精度上均与概率图模型的算法不相上下，而产生的模型要小得多。后面的章节将给出具体的实现。

9.2.2 Word2Vec 的算法原理

毫无疑问，Word2Vec 属于一种神经网络架构的概率语言模型，对于任何一个神经网络的架构，都要从它的网络结构开始研究：

1. Word2Vec 神经网络架构

图 9.8 所示为 Word2Vec 神经网络架构，包括：输入层、投影层、隐含层和输出层。其中， W 表示投影层与隐含层之间的权值矩阵； U 表示隐含层与输出层之间的权值矩阵； P 表示隐含层上的偏置向量； q 表示输出层上的偏置向量。

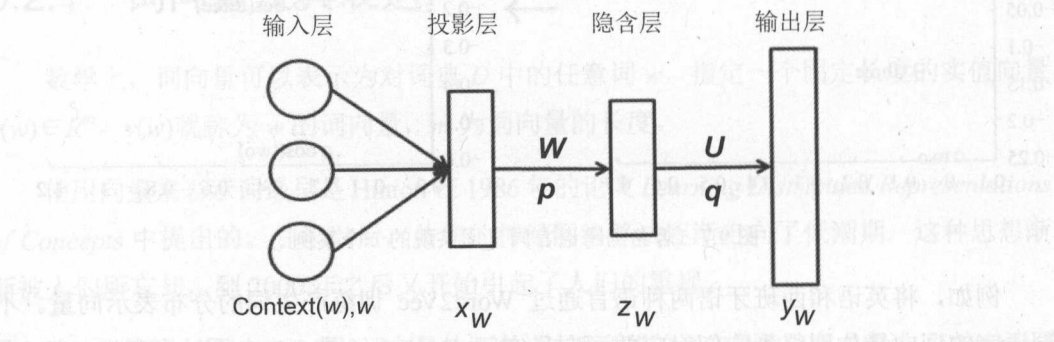


图 9.8 Word2Vec 神经网络架构

设输入语料为 C ，词向量长度为 m ，这些都从外部给出。从 C 中逐个遍历每一个词 w ，设 n 为 w 的上下文长度，则 $\text{Context}(w)$ 就取为其前面的 $n-1$ 个词（类似于 $n\text{-gram}$ ），这样 $(\text{Context}(w), w)$ 就构成了一个二元对，这个二元对就构成了一个训练样本。

语料 C 和词向量长度 m 给定后，投影层和输出层的规模就确定了：输入层包含了 $\text{Context}(w)$ 中 $n-1$ 个词的词向量，而投影层的向量 x_w 是这样构造的：将输入层的 $n-1$ 个词向量按顺序首尾相接地拼起来，形成一个长向量，其长度当然就是 $(n-1)m$ 。也就是说，每个词的向量长度都为 m ，共有 $n-1$ 个词，因此投影层为 $(n-1)m$ 。

输出层是一棵 Huffman 树，每个词汇是这棵树中的一个叶子节点。因此，输出层的维度为 $N_o=|D|$ ，即语料 C 中所有的词汇量数目。

隐藏层的规模 N_h 是可调参数，由用户从外部指定。

接下来，讨论样本 $(\text{Context}(w), w)$ 经过上述神经网络时是如何参与运算的。

$$\begin{cases} z_w = \tanh(Wx_w + p) \\ y_w = Uz_w + q \\ p(w|\text{Context}(w)) = \text{Softmax}(y_w) \end{cases}$$

第一式为 \tanh 双曲正切函数，用作隐藏层的激活函数。 \tanh 作用在向量上表示它作用在向量的每一个分量上。

第二式计算得到的 $y_w = (y_{w,1}, y_{w,2}, \dots, y_{w,N})$ 只是一个长度为 N 的向量。由于 y_w 的分量的计算结果不能在 $0 \sim 1$ 之间，不能表示某个词 w 上下文为 $\text{Context}(w)$ 的形式。

第三式通过 softmax 函数计算得到 $p(w | \text{Context}(w))$ ，下一个词恰为词典 D 中第 i 个词的概率。

这里简单说一下 softmax 函数。

$$p(w | \text{Context}(w)) = \text{Softmax}(y_w) = \frac{\exp(y_{w,i})}{\sum_{i=1}^N \exp(y_{w,i})}$$

其中， i 表示词 w 在词典 D 中的索引。 Softmax 是前文讲过的 Logistic 函数对于多分类情况的一个扩展。前文讲过 Logistic 可以用于二分类的情况，而在多分类的情况下深度学习中最常使用的函数是 Softmax 。

需要注意的是，上式的最后一项，一般的神经网络输入都是已知的，而 Word2Vec 中的输入是通过 Softmax 计算才得到的。

2. CBOW (Continuous Bag-of-Words Model) 模型

CBOW 是 Word2Vec 最重要的模型，输入是周围词的词向量，而输出是当前词的词向量。也就是说，通过周围的词来预测当前词。CBOW 模型如图 9.9 所示。

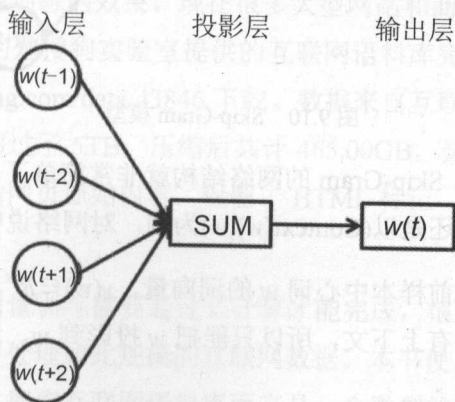


图 9.9 CBOW 模型

CBOW 网络包括：输入层、投影层和输出层。还是以 $(\text{Context}(w), w)$ 为例。 $\text{Context}(w)$ 为 w 的上下文，单侧个数为 c ，总长度为 $2c$ 。对网络说明如下。

- ❑ 输入层。Context(w)中 $2c$ 个词的词向量 $v(\text{Context}(w)_1), v(\text{Context}(w)_2), \dots, v(\text{Context}(w)_{2c}) \in R^m$ 。这里, m 定义为词向量的长度。
- ❑ 投影层: 将输入层的 $2c$ 个向量做求和累加, 即 $x_w = \sum_{i=1}^{2c} v(\text{Context}(w)_i)$
- ❑ 输出层。输出层对应着一棵 Huffman 树, 它是以语料中出现过的词当叶子节点, 以各词在语料中出现的次数为权值构造出来的 Huffman 树。这棵树中, 叶子节点共 $N (=|D|, D \text{ 为词典的大小})$ 个, 非叶子节点为 $N-1$ 个。

这里需要说明的是, CBOW 模型中从输入层到投影层的操作是累加求和, 而不是拼接。而且, CBOW 中没有隐藏层, 投影层直接连接着输出层。

3. Skip-Gram (Continuous Skip-Gram Model) 模型

Skip-Gram 与 CBOW 正相反, 输入是当前词的词向量, 而输出是周围词的词向量。也就是说, 通过当前词来预测周围的词。Skip-Gram 模型如图 9.10 所示。

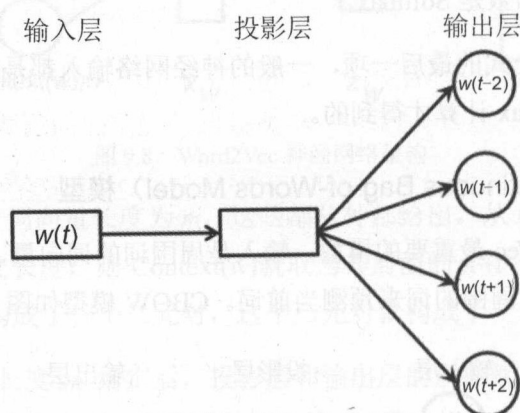


图 9.10 Skip-Gram 模型

理解了 CBOW 模型, Skip-Gram 的网络结构就非常简单了。它也包括三个层次: 输入层、投影层和输出层。还是以 $(\text{Context}(w), w)$ 为例, 对网络说明如下。

- ❑ 输入层。只含当前样本中心词 w 的词向量, $v(w) \in R^m$ 。
- ❑ 投影层。因为没有上下文, 所以只能把 w 投影到 w , 这是一个恒等投影、多余步骤。
- ❑ 输出层。对应着一棵 Huffman 树。

有关两个网络的优化和推导过程, 本节就不给出了。读者可详细阅读《Word2Vec 中的数学原理详解》一文, 该文对网络的每一个细节都给出了重要的说明。同时, 读者

可结合《Word2Vec 源码解析》一文，了解 Word2Vec 实现的每个细节。毕竟 NLP 不仅仅是数学。

9.2.3 训练词向量

目前常用的 Word2Vec 实现共有三个版本，最著名的是 Google 发布的用 C 语言编写的 Word2Vec，除从 Google 官网下载之外，读者也可以从 <http://www.threedweb.cn/thread-1596-1-1.html> 下载最新版本的源码。历经三年的时间，此版本已经进行了多次升级，支持多种结构（词汇、短语等）的向量化计算。唯一不足的是该版本只能运行在 Linux 环境中。

第二个是基于 Python 的 Gensim 框架 (<https://radimrehurek.com/gensim/index.html>)。该框架不仅将 Word2Vec 与 Python 做了整合，这使得 Word2Vec 的应用扩展到了 Windows 平台，而且，Gensim 还提供了基于 LSA、LDA、HDP 的主体框架。该框架希望涵盖词向量表达的所有算法系统。

第三个是基于 Java 的 Word2Vec 实现 (https://github.com/NLPchina/Word2VEC_java)。该框架有助于读者学习 Word2Vec 的源码。常用于小型语料的测试。

本节使用的词向量工具是 Google 公司发布的用 C 语言编写的 Word2Vec。下面的章节会用到 Python 版本的 Gensim 框架。

使用 Word2Vec 进行训练，一般需要大规模的语料（5GB 以上），这些语料还需要精准的分词，最终才能体现训练的效果。现在很多大型网站和商业机构都提供相关语料的下载。其中，比较著名的是搜狗实验室提供的互联网语料库完整版（SogouT）。该语料库可从 <http://www.datatang.com/data/43846> 下载。数据来自互联网各种类型的 1.3 亿个原始网页，压缩前的大小超过了 5TB，压缩后共计 465.00GB。数据版本为 2008 年完整的网页历史版。数据格式为网页原始内容，保留了 HTML 标记、页面 ID、页面 URL 和页面原始内容。

这种语料规模的词向量训练需要通过云计算才能完成，很难想象一般性的研究机构或商业公司能在短时间内处理如此规模的互联网数据。本书使用的是微软的 MSR 分词语料库，规模不大，相比搜狗互联网语料库而言是一个微型的语料库，因为这个阶段的训练结果在后面的 LSTM 序列标注阶段要使用，所以训练过程仅作为演示使用，但即便这么小规模语料仍然显示出算法在相似度间的显著效果。

给出如下命令行。

```
./word2vec -train msr.utf8.txt -output vectors.bin -cbow 0 -size 200 -window 5
-negative 0 -hs 1 -sample 1e-3 -threads 12 -binary 1
```

命令行参数说明如下。

```
-train <file> : 需要训练成模型的文件名 (<file>)
-output <file>: 保存为词向量的文件名 (<file>)
-size <int>: 词向量的维度, 默认是 100
-window <int>: 上下文窗口的大小, 默认是 5
-sample <float>: 设置词频采样的阈值, 在训练数据中高于此阈值的词汇会被欠采样
(down-sampled), 它是一个经验值, 忽视掉频率过高的词的参数; 默认是 1e-3, 可用范围为 (0, 1e-5)。
-hs <int>: 使用层次 (Hierarchical) Softmax (对罕见字有利), 默认是 0,
-negative <int>: 负采样数量 (对常见词和低维向量有利); 默认是 5, 常用值为 3~10 (不能
为 0)
-threads <int>: 线程数 (默认是 12)
-iter <int>: 迭代次数 (默认是 5)
-min-count <int>: 少于此词频的词汇会被忽略; 默认是 5
-alpha <float>: 学习参数, skip-gram 默认是 0.025; CBOW 默认是 0.05
-classes <int>: 设置词汇的聚类个数
-debug <int>: 设置 debug 模式
-binary <int>: 存储词汇的结果向量为二进制模式, 默认是 0 (关闭)
-save-vocab <file>: 存储学习出的词向量文件 (<file>)
-read-vocab <file>: 读取已经学习出的词向量文件 (<file>)
-cbow <int>: 设置是否使用 cbow 模型; 默认是 1 (0 为 skip-gram 模型)
```

输出结果可以有两种: 一种是 txt 格式的词向量文件; 另一种是二进制的文件。打开 txt 格式的文件, 部分内容显示如下。

```
经济 0.349525 -0.231814 0.246029 -0.010521 0.276938 -0.051195 -0.012460 0.346636 -0.024882 0.238553 0.49101
中国 -0.012340 -0.133595 0.506861 0.242898 -0.038924 -0.121401 -0.232330 0.127766 -0.293897 -0.000815 0.208
从 -0.215471 -0.349014 0.073415 0.018083 0.009230 -0.073501 -0.113572 0.043045 -0.347602 -0.172102 0.407675
我 -0.157422 -0.239220 -0.276118 0.091978 0.123269 0.181570 -0.401933 0.070169 -0.299026 0.578140 -0.156384
我们 0.019763 -0.074935 -0.001470 -0.129272 -0.005197 0.161525 -0.317441 0.190774 -0.227338 0.086228 -0.034
工作 0.021017 -0.177481 0.249019 -0.281438 0.545852 0.170903 0.066149 0.285722 -0.168277 -0.171851 0.085887
企业 0.101199 -0.487773 0.081173 0.128727 0.308858 -0.232959 0.006138 0.569418 0.073654 -0.145506 -0.495158
- -0.486756 -0.523793 0.862641 0.476276 0.100009 0.499068 0.777304 0.377777 0.308615 0.167404 -0.304859 0.0
问题 0.055623 0.038131 0.189265 0.041917 -0.333804 -0.518506 -0.114580 0.397354 0.232388 -0.259705 -0.11221
大 -0.172353 -0.223649 0.128173 -0.166425 0.344379 0.262095 0.102873 0.460137 0.180254 0.074527 -0.172461 -
```

使用二进制的结果展示词汇间的相似度, 代码如下。

```
./distance vector.bin
```

输入词汇: 文化。

图 9.11 所示为“文化”语义相似词, 是 Word2Vec 产生的结果。

```
Enter word or sentence (EXIT to break): 文化
Word: 文化 Position in vocabulary: 152
```

Word	Cosine distance
博大精深	0.839596
人文	0.796721
传统	0.736364
民间艺术	0.735981
体育运动	0.725998
礼仪	0.719927
熏陶	0.709557
风土人情	0.708395
瑰宝	0.706779
遗产	0.703613
美德	0.701274
品位	0.696130
民族文化	0.693234
积淀	0.685298
伦理	0.683027
儒家	0.682990
东西方	0.678683
精华	0.677927

图 9.11 “文化”语义相似词

训练大规模语料是一个既耗时又大量占用 CPU 资源的过程。很多研究机构为了迅速获得实验的结果，常使用已经训练好的词向量。主要原因还由于对大规模语料的预处理依赖一个同样大规模、高精度、高效率的中文分词器，否则很难达到预想的训练效果。如果你不打算自己训练词向量模型，那么也可以从网上找到相应的词向量模型。现在常用的词向量模型分为收费和免费两大类。即使是收费其价格也不贵，并附带项目的调用程序，内容如下。

(1) 相似词、同义词知识库（基于 30 亿条微博数据，包含调用程序——基于 Word2Vec）。读者可从 <http://www.datatang.com/data/46677> 下载。

基于 30 亿条微博数据计算获取的 Word2Vec 模型，由于训练语料库较大，使得获取到的相似词更精准、涵盖更广，并附带 C 语言程序，可以非常方便地批量获取指定词的相似词。所获得的结果格式是“词：权重”。

数据大小：755.38MB。

(2) 相似词知识库，Word2Vec 模型，基于 5GB 维基百科中文语料。读者可从 <http://www.datatang.com/data/47233> 下载。

内含 Word2Vec 源码、维基百科中文语料 Python 抽取器、词向量模型，以及基于词向量模型计算词语相似度的 Java 代码，通过 Java 工程可以批量计算词语组相似度。词向

量模型是基于 5GB 多的维基百科中文语料训练出来的，精度可以得到保证。训练前已经用 OpenCC 进行了繁简转换。

数据大小：616.26MB。

如果你希望找到可以免费使用的词向量模型数据，可能规模要小一些，建议你从 <http://txt2vec.codeplex.com/> 下载到二进制包。网站给出的案例如下。

手串 1

佛珠 0.918781571749997
小叶紫檀 0.897870467450521
手钏 0.868526208199693
菩提子 0.85667693515943
紫檀 0.855529437116288
佛珠手链 0.849541378712106
雕件 0.847901026881494
砗磲 0.842016069107114
小叶檀 0.839194380950776
星月菩提子 0.838186634277951
檀香木 0.837212392914782
沉香木 0.83575322205817
星月菩提 0.83494878072285
黄花梨 0.831824567567293
平安扣 0.831679080640205
紫檀木 0.830029415546653
小叶紫檀手串 0.82838028045219
原籽 0.823008017930358
.....

上述文件是以二进制形式保存的，需要使用 <https://github.com/zhongkaifu/RNNSharp> 解析。这是一个基于 CRF-LSTM 的高精度命名实体识别工具。项目使用 C 语言编写，代码并不难读，总词汇数将近 700 000 个词，生成的词向量数据为 200 维，部分解析出的结果如下。

171423 中毒症状:-0.0679101118419207,0.110328259152416,0.0137756968801185,-0.0348270993948245,-0.0583684051156343
171424 谭小:-0.0534902053477119,0.0464330911194762,-0.0183293356252258,-0.0532591191012103,-0.0107785172730798,-
171425 慢性乙肝:-0.0487475106337111,-0.00271269462625653,0.0218041256553617,-0.0449241373531417,-0.0540266244565
171426 可全:-0.00549155563563541,-0.0275969267156586,0.0224655243435983,-0.08119136807912,0.0750455583233297,-0.0
171427 有点慢:-0.0237127838580224,0.0789343186054692,0.0871438083758978,0.0227480879357494,-0.0731056002152947,0
171428 卡贝:-0.0137953471904155,0.0532687579586589,-0.0746823645920314,0.0931016469091341,-0.158523898869019,-0.
171429 lube:-0.00435803838047436,0.0307130170460142,-0.0248226402230253,0.055110711788508,0.00596268597142137,-0.
171430 内部控制评价:-0.0859716888572665,0.0418870769381197,-0.0395342191624064,-0.0797827308110835,0.04859989214
171431 胡南:-0.0294011275804542,-0.0131909795733655,0.0142504806670995,-0.0400984218415534,0.083292622122674,-0.0
171432 工中:-0.125976880987866,0.0663218096464172,-0.111282156394687,-0.0927891415003592,0.0110874513741598,-0.02
171433 乡村爱情交响曲:-0.00948165007322863,-0.0129460831465235,0.0665049464136254,0.0630199399733496,0.060945971
171434 偿债:-0.184264193479147,-0.0454079592112574,0.0884090614526706,-0.055862511203982,0.0491774592904879,-0.0
171435 星房:-0.0253511119835126,0.0492873002777777,-0.0891219990233053,-0.0654331310105845,0.031323762251228,-0.
171436 对镜镜子:-0.114151005898875,0.04204186020875,-0.0123578424185482,0.0386168945905869,0.167988953458038,0.02
171437 清辉:-0.016771983048694,-0.0921373238985196,-0.0951292745338764,-0.0632661293846955,-0.0656063338373958,-0
171438 sift:-0.028642361072066,-0.0682449316188052,-0.0662554209584602,-0.0343067998515453,0.0781107221701765,-0.
171439 鸠江区:-0.026528740095362,0.0338996808160015,-0.0425807400341551,0.00494636807469875,0.0128905388454638,-

解析后总文件大小为 2.5GB，压缩成一个大小为 1.08GB 的文件包，文件名为 wordembedding.rar。读者可从 <http://www.threedweb.cn/thread-1597-1-1.html> 下载。

9.2.4 大规模上下位关系的自动识别

第 7 章详细分析过 HowNet 知识库的架构，该知识库由董振东先生主持建设，共历经十年，才最终完成。由此可见，人工构建知识库是一种知识密集且费时的的工作。然而，类似 HowNet 和 WordNet 这些语义资源却在实际使用中都极大地受限于它们覆盖的领域，效果并不好。因此，许多研究人员多年来都不断尝试自动提取语义关系或构建分类法。表 9.1 显示了有关研究的执行效果。

表 9.1 Word2Vec之前的上下位关系识别方法

	P(%)	R(%)	F(%)
Mwiki+CilinE	92.41	60.61	73.20
Mpattern	97.47	21.41	35.11
Msnow	60.88	25.67	36.11
MbalApinc	54.96	53.38	54.16
MinvCL	49.63	62.84	55.46
Mfu	87.40	48.19	62.13

MWiki+CilinE 是由 Suchanek 等人（2008 年）提出的基于人工构建的层次可拓方法。在中文处理中，常用的方法是利用中文维基百科（Wikipedia）的类别分类扩展 CilinE（哈工大同义词词林：<http://www.ltp-cloud.com/download/>）。如表 9.1 所示，该方法虽然实现了高精度，但召回较低，主要是因为维基百科语料的覆盖范围有限。

中文赫斯特风格词汇模式如表 9.2 所示。

表 9.2 中文赫斯特风格词汇模式

No	Pattern
1	w 是[一个 一种] h
2	w[、]等h
3	h[,]叫作w
4	h[,]像]如w
5	h[,]特别是w

MPattern 是由赫斯特提出的基于模式的方法。在中文处理中，常用的方法是从中文百科语料库中提取上下位关系，百科语料库也被用于训练字嵌入。这里使用中文赫斯特模

式（见表 9.2），其中， w 表示一个字，和 h 代表其一个上位词。结果表明，仅有一小部分上位词基于这些模式，因为只有少数上位词关系使用这种固定的模式表示，大量上下位关系被表示为更加灵活的形式。

MSnow 方法最初是由 Snow 等人提出的（2005 年）。从 CilinE 学习同样的训练数据用作种子上下位词对。词典式一句法模式使用种子从百科语料库提取。然后，开发一个 Logistic 回归分类来识别上下位关系。此方法必须依赖于一个精准的句法分析器，并且自动提取模式的质量难以保证。

MbalApinc（Kotlerman 等，2010 年）和 MinvCL（Lenci 和 Benotto，2012 年）很相似，它们将每个词汇表示为一个特征向量，其中每个维度是词汇的一个 PMI 值及其上下文。通过计算每个词汇对的评分，并应用一个阈值，以确定它是否为一个上下位关系。

Mfu 是由 Fu 等在 2013 年设计的 Web 数据挖掘方法。此方法能够挖掘出来自多种来源的给定词汇 w 的上位词列表，并返回上位词的层次。通过从上位词列表中，选择超过每个词阈值之上的含有评分的上位词。此方法假定，多个来源的经常共现的名词或名词短语 n ，包含着 w ，则表明 n 有可能是 w 的上位词。如果 w 是一个实体，则这种方法效果很好。但如果是一个常用的语义概念，则效果不佳。主要的原因可能是，语料中有相当多关于实体介绍的页面，它们不是在网络中的常见的词汇。

由此可见，上述各种方法都存在着缺陷。这个问题是由哈工大的刘挺为首的《大词林》（URL: <http://www.bigcilin.com/browser/>）项目组最终解决的。下面介绍他们实现自动上下位关系抽取的全过程。

经过大量的研究，他们观察到，词嵌入通过捕捉大量的句法/语义关系，保留了很多语言规律。一个著名的例子： $v(\text{king}) - v(\text{queen}) \approx v(\text{man}) - v(\text{woman})$ ，表明词嵌入向量的差值确实代表两个词对之间的某种语义关系。

需要搞清楚的是，是否这种情况也适用于上下位关系。他们设计了一个简单的实验，使用一些随机抽取的汉语中的上下位词对，计算嵌入向量的差值以测量它们之间的相似性。

结果如表 9.3 所示。第一个例子表明一个词也可以利用 Word 向量的差值映射出其上位词。然而，从“木匠”到“工人”比从“金鱼”到“鱼”具有更大偏差，表明上下义关系应该比单独矢量的偏差更复杂。为了验证这个假设，这里计算了在训练集中所有的上下义关系词对的嵌入向量间的偏差，并使之可视化。通过考察图 9.12 训练集数据中关系聚类的分布，发现上下义关系可以分解为更细粒度的关系。此外，动物的关系在空间上更密切，而人的职业在空间上更分散。

表 9.3 在上下位词对中的词嵌入的偏移计算

No	Examples
1	v (虾) -v (对虾) ≈ v (鱼) -v (金鱼)
2	v (工人) -v (木匠) ≈ v (演员) -v (小丑)
3	v (工人) -v (木匠) ≈ v (鱼) -v (金鱼)

为了应对这一挑战，我们提出了使用投影矩阵学习的上下义关系。

1. 均匀线性映射

直观地说，假设所有的词可以基于统一的转移矩阵被映射到其上位词。即，给定一个词 x 和其上位词 y ，存在一个矩阵 Φ ，从而 $y=\Phi x$ 。为简单起见，使用与词相同的符号来表示嵌入向量。获得用于所有上下位词对的投影一致的精确 Φ 是困难的。取而代之的是，可以使用训练数据学习一个近似的 Φ ，使用式 (1)，最小化均方误差。

$$\Phi^* = \arg \min_{\Phi} \frac{1}{N} \sum_{(x,y)} \|\Phi x - y\|^2 \tag{1}$$

其中， N 是 (x,y) 的词汇对在训练数据的数量。这是一个典型的线性回归问题。唯一不同的是，我们的预测是多维向量，而不是标量值。使用随机梯度下降法进行优化。如图 9.12 所示，向量偏差分布在一些集群中。图 9.12 左图显示了关于动物的上下位关系。图 9.12 右图显示了关于人们的职业的关系。

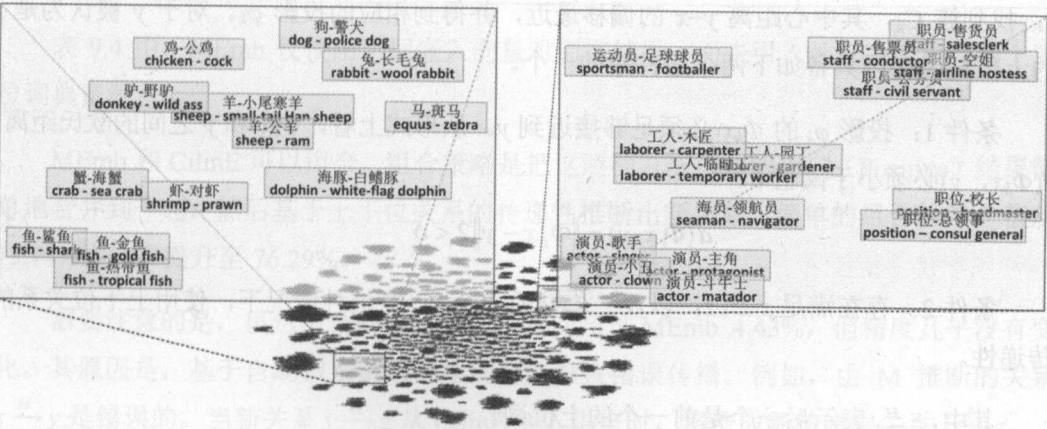


图 9.12 训练数据中向量偏差值的聚类

2. 分段线性映射

均匀线性映射可能对拟合所有的上下位词对仍有欠表达的情况。这是因为上下位关

系是相当多样的。为了更好地对各种上位词一下义关系建模，这里应用分段线性回归的方法。

首先，输入空间被分割为多个区域。即所有词对 (x, y) 中的训练数据首先聚成几个簇，其中每个簇中的单词对的映射表现出相似的上下位关系。每个词对 (x, y) 被表示为其向量的偏差： $y - x$ 。然后执行聚类，这么做的原因如下：（1）Mikolov 的工作已经表明，向量偏差暗示出固定水平的语义关系。（2）使用向量的偏差进行聚类效果更加良好，越接近的词对代表的关系更相似，如图 9.12 所示。然后对每个簇分别学习一个映射，内容如下。

$$\Phi_k^* = \arg \min_{\Phi_k} \frac{1}{N_k} \sum_{(x, y) \in C_k} \|\Phi_k x - y\|^2 \quad (2)$$

其中， N_k 为第 k 个簇 C_k 的词对数量。使用 K-means 聚类算法， k 是在开发数据集进行调整的。

3. 识别上下位关系

要学习的这个映射矩阵，他们使用哈工大汉语同义词林（扩展版）（CilinE 的简称），其中包括 100 093 个词提取的训练数据（车等，2010 年）。CilinE 组织为 5 个层次，其中的每个词都被上下位关系联系在一起构成层次结构。在 CilinE 每个词都有一个或多个语义代码（有些词汇是多义），指示其层次结构中的位置。

给定训练数据和相应的投影，可以找出两个词是否有上下位关系。给定两个词 x 和 y ，找到簇 C_k ，其中心距离 $y - x$ 的偏移最近，并得到相应的投影 Φ_k 。对于 y 被认为是 x 的上位词，必须具备如下两个条件中的一个。

条件 1：投影 Φ_k 的 $\Phi_k x$ 必须足够接近到 y 。从形式上看， $\Phi_k x$ 和 y 之间的欧氏距离： $d(\Phi_k x, y)$ 必须小于阈值 δ 。

$$d(\Phi_k x, y) = \|\Phi_k x - y\|_2 < \delta \quad (3)$$

条件 2：存在满足 $x \xrightarrow{H} z$ ，以及 $z \xrightarrow{H} y$ 的一个 z 。在这种情况下，使用上下位关系的传递性。

其中， \xrightarrow{H} 表示后一个是前一个的上位词。

此外，最终的层次应该是一个 DAG（有向无环图）。然而，映射的方法在理论上不能保证，由于映射来自成对的上下位关系，而对整个层次结构并不了解。所有成对上下位关系识别方法都将会遇到这个问题。这是一个有趣的问题，如何构建一个符合 DAG 形式的全局最优的语义层次结构呢？这不是本文的重点。这里可以给出一些简要的说明，

如果一些冲突发生，即存在关系圈，那么试探性去除或颠倒最弱路径（见图 9.13）。如果一个圆只有两个节点，那么删除最弱的路径。如果一个圆有两个以上的节点，那么颠倒构成间接上下位关系的最弱路径的方向。

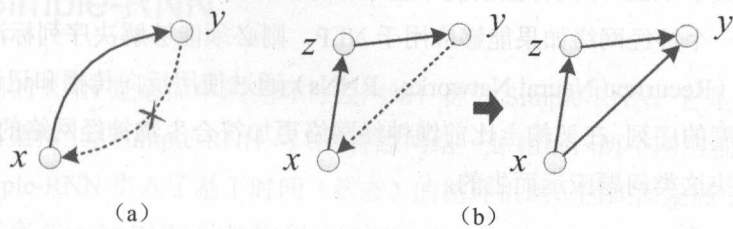


图 9.13 三种路径去除或翻转

如图 9.13 (a) 所示，如果 $d(\Phi_{y,x}) > d(\Phi_{x,y})$ ，那么移除从 y 到 x 的路径；如图 9.13 (b) 所示，如果 $d(\Phi_{y,x}) > d(\Phi_{x,z})$ 并且 $d(\Phi_{y,x}) > d(\Phi_{z,y})$ ，那么翻转从 y 到 x 的路径方向。

经过这样的训练过程，他们设计了三种模型，并给出了三种模型的训练结果，如表 9.4 所示。

表 9.4 Word2Vec 之后的上下位关系识别效果

	$P(\%)$	$R(\%)$	$F(\%)$
MEmb	80.54	67.99	73.74
MEmb+CilinE	80.59	72.42	76.29
M Emb+wiki+CilinE	79.78	80.81	80.29

表 9.4 中，MEmb 仅使用了词嵌入向量对训练结果，而未引入外部手工构建的上下位词典资源。

MEmb 和 CilinE 可以组合。组合策略是把这两种方法得到的正向 (Positive) 结果简单地合并到一起，然后基于上下位关系的传递性推断出新关系。简单的组合就使 F 值从 73.74% 进一步提升至 76.29%。

需要注意的是，虽然组合的方法实现的召回超过 MEmb 4.43%，但精度几乎没有变化。其原因是，基于自动识别关系的推理可能导致错误传播。例如，由 M 推断的关系 $x \xrightarrow{H} y$ 是错误的。当新关系 $y \xrightarrow{H} z$ 从 CilinE 加入进来时，将造成新的错误关系 $x \xrightarrow{H} z$ 。

MEmb 与 Wiki+CilinE 结合比基准的 Wiki+CilinE 在 F 值上提升了 7%。因此，MEmb 方法与手动构建层次扩展方法的互补结果是最佳的选择。

MEmb 是基于词嵌入入了该方法。如表 9.4 所示，该方法实现了比所有以前方法都好得多的召回率和 $F1$ 值。它超过之前的所有的方法成为最终的选择。

9.3 NLP 与 RNN

前面提到关于 NLP 序列标注的几个基本任务，如分词、词性标注、未登录词识别、语义组块等。一个神经网络如果能够应用于 NLP，则必须能够解决序列标注的问题。而循环神经网络（Recurrent Neural Networks, RNNs）通过使用反向传播和记忆的机制，能够处理任意长度的序列，在架构上比前馈神经网络更加符合生物神经网络的结构。因此，其正是为了解决这类问题应运而生的。

RNN 及改进的 LSTM 等深度学习模型都是基于神经网络而发展起来的认知计算模型。从原理来看，它们都源于认知语言学中的“顺序象似性”原理：文字符号与其上下文构成了一个“像”，这个“像”可以被认为是符号与符号的组合——词汇，也可以被认为是词汇与词汇的句法关系——依存关系。算法的训练过程，是通过正向和反馈两个过程从训练语料中学习出识别这些“像”的能力，并记录下识别“像”的模型数据，当输入新的句子时，算法可以利用存储的模型数据识别出新输入中类似的“像”。

如图 9.14 所示，共分为 5 部分，从左到右，第一幅图是一个输入（单一标签）对应一个输出（单一标签），即“one to one”方式；第二幅图为一个输入对应多个输出，即“one to many”方式，这种网络架构广泛用于图片的对象识别领域，即输入的是一张图片，输出的是一个文本序列；第三幅图为多个输入对应一个输出，即“many to one”方式，这种网络架构广泛用于文本分类或视频片段分类的任务，输入的是视频或文本集合，输出的是类别标签；第四幅图是多个输入对应有间隔的多个输出，即第一种“many to many”方式，这种网络架构就是机器翻译系统，输入的是一种语言的文本，输出的是另一种语言的文本；第五幅图是多个输入严格对应多个输出，即第二种“many to many”方式，这种网络架构就是 NLP 中广泛使用的序列标注任务。

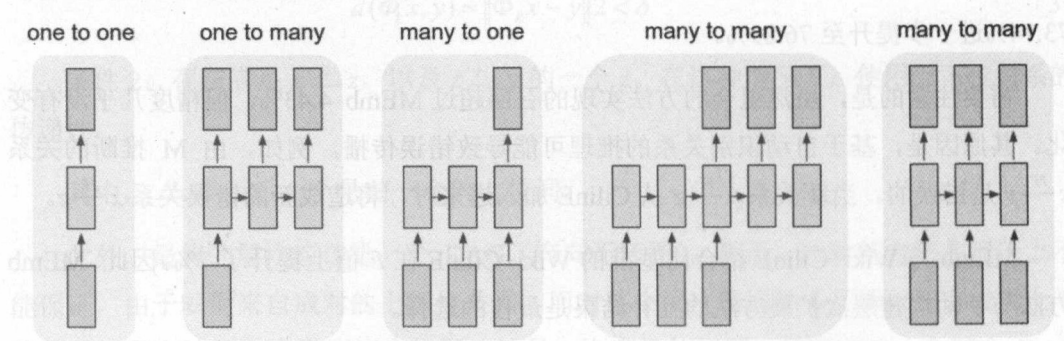


图 9.14 5 种不同的 RNN 架构

在众多的深度学习网络中，循环神经网络由于能够接收序列输入，也能得到序列输出，在自然语言处理中取得了巨大成功，并得到广泛应用。

9.3.1 Simple-RNN

本节介绍的 RNN 是最简单的循环神经网络，称为 Simple-RNN，它是后面 LSTM 的基础。从网络架构上，Simple-RNN 与 BP 神经网络一脉相承，两个网络都有前馈层和反馈层，但 Simple-RNN 引入了基于时间（状态）的循环机制，因此而发展了 BP 网络。下面从整体上考察 Simple-RNN 的架构和训练运行。

图 9.15 所示为 Simple-RNN 的神经网络示意图。神经网络为 A，通过读取某个时间（状态）的输入 x_t ，然后输出一个值 h_t 。循环可以使得信息从当前时间步传递到下一时间步。

这些循环使得 RNN 可以被看作同一网络在不同时间步的多次循环，每个神经元会把更新的结果传递给下一时间步。为了更清楚地说明，将这个循环展开，放大图 9.15 中的神经网络 A，考察一下网络的细节。

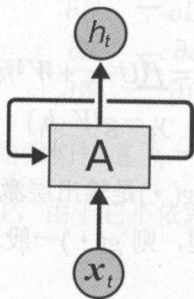


图 9.15 Simple-RNN 的神经网络示意图

如图 9.16 所示，网络某一时刻的输入 x_t ，与之前介绍的 BP 神经网络的输入一样， x_t 是一个 n 维向量，不同的是，递归网络的输入将是一整个序列，也就是 $\mathbf{x}=[x_0, \dots, x_{t-1}, x_t, x_{t+1}, \dots, x_T]$ ，对于语言模型，每一个 x_t 将代表一个词向量，一整个序列就代表一句话。 h_t 代表时刻 t 的隐含状态， y_t 代表时刻 t 的输出。

- ❑ 输入层到隐含层直接的权重由 U 表示，它将原始输入进行抽象作为隐含层的输入。
- ❑ 隐含层到隐含层的权重 W ，它是网络的记忆控制者，负责调度记忆。
- ❑ 隐含层到输出层的权重 V ，从隐含层学习到的表示将通过它再一次抽象，并作为最终输出。

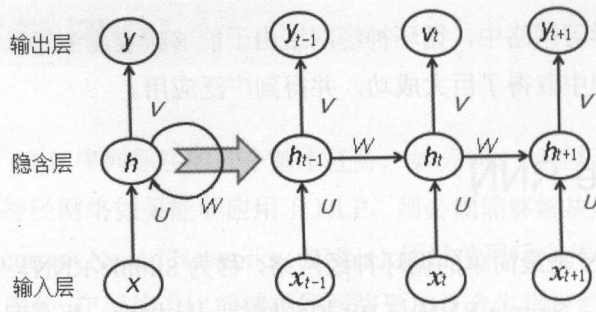


图 9.16 RNN 神经元在不同时间步的传递

将 RNN 展开之后，似乎一切都很明了，前向传播（Forward Propagation）依次按照时间的顺序计算一次即可，反向传播（Back Propagation）从最后一个时间将累积的残差传递回来即可，跟普通的 BP 神经网络训练并没有本质上的不同。由于加入了时间顺序，计算的方式有所不同，这称为 BPTT（Back Propagation Through Time）算法。该算法的细节如下。

1. 前向传播

首先在 $t=0$ 的时刻， U 、 V 、 W 都被随机初始化好， h_0 通常初始化为 0，然后进行如下计算。

$$h_1 = f(U \cdot x_1 + W \cdot h_0)$$
$$y_1 = g(V \cdot h_1)$$

其中， $f(\cdot)$ 是隐含层激活函数， $g(\cdot)$ 是输出层激活函数。一般， $f(\cdot)$ 可以选择 tanh、relu、logistic 等；如果序列标注问题，则 $g(\cdot)$ 一般选择 Softmax。 h_1 为第一层的输出， y_1 为预测的结果。

这样，当在时间步 t 时，公式变为如下的形式。

$$h_t = f(Ux_t + Wh_{t-1})$$
$$y_t = g(Vh_t)$$

需要注意的是，循环神经网络是拥有记忆能力的，而这种能力是通过 W 记录了以往的输入状态，作为下次的辅助输入。隐藏状态可以理解为如下内容。

$$h = f(\text{现有的输入} + \text{过去记忆})$$

其中，全局误差如下。

$$E = \sum_t e = \sum_{i=1}^t f_e(y_i - d_i)$$

这里 E 是全局误差, e_i 是第 i 个时间步长的误差, y 是输出层预测结果, d 是实际结果。误差函数 f_e 的选择可以是交叉熵 (Cross Entropy), 也可以是平方误差项等。

2. 反向传播

这与之前讲解的 BP 神经网络用到的反向传播方法的思想是一致的, 也就是利用输出层的误差 e (Cost Function), 求解各个权重的梯度 ∇V 、 ∇U 、 ∇W , 然后利用梯度下降法更新各个权重。

求解各个权重的递推公式如下。

$$V(t+1) = V(t) + \alpha \cdot \nabla V$$

$$U(t+1) = U(t) + \alpha \cdot \nabla U$$

$$W(t+1) = W(t) + \alpha \cdot \nabla W$$

现在的问题就是如何求解各个权重的梯度。即我们的目标就是要求取:

$$\nabla V = \frac{\partial E}{\partial V} = \sum_i \frac{\partial e_i}{\partial V}$$

$$\nabla U = \frac{\partial E}{\partial U} = \sum_i \frac{\partial e_i}{\partial U}$$

$$\nabla W = \frac{\partial E}{\partial W} = \sum_i \frac{\partial e_i}{\partial W}$$

求解的顺序分为如下两步, 首先我们知道 $h_t = f(Ux_t + Wh_{t-1})$, 对于任何代价函数, 直接求取每一时刻的 $\sum_i \frac{\partial e_i}{\partial V}$, 得到 ∇V , 由于它不依赖之前的状态, 可以直接求导获得。

然后简单加和即可。

$$\nabla V = \sum_i e_i \cdot h_i$$

但是 ∇U 、 ∇W 依赖于之前的状态, 不能直接求导, 需要定义中间变量:

$$\delta = \frac{\partial e_i}{\partial h_i}$$

首先计算出输出层的 δ^y , 再向后传播至各层 δ^h , 依次类推, 直至输入层:

$$\delta_t^h = (W^T \delta_{t+1}^h + V^T \delta_t^y) \cdot f'(h_t)$$

这里 $*$ 表示点积, 只要计算出 δ^y , 以及后面所有的 δ^h , 即可通过如下计算出 ∇U 、 ∇W 。

$$\nabla W = \sum_t \delta_t^h \times h_t$$

$$\nabla U = \sum_t \delta_t^h \times x_t$$

为了便于读者进一步理解公式，这里给出 Simple-RNN 的部分源代码，内容如下。

#3. 主程序--训练过程:

```
for j in xrange(maxiter):
```

```
    # 在实际应用中，可以从训练集中查询到一个样本：生成形如[a] [b]--> [c]的样本:
```

```
    a,a_int,b,b_int,c,c_int = gensample(dataset,largest_number)
```

```
    # 初始化一个空的二进制数组，用来存储神经网络的预测值
```

```
    d = np.zeros_like(c)
```

```
    overallError = 0 # 重置全局误差
```

```
    layer_2_deltas = list(); # 记录 layer 2 的导数值
```

```
    layer_1_values = list(); # layer 1 的值。
```

```
    layer_1_values.append(np.zeros(hidden_dim)) # 初始化时无值，存储一个全零的向量
```

```
    # 正向传播过程：逐个bit位(0,1)的遍历二进制数字。
```

```
    for position in xrange(binary_dim):
```

```
        indx = binary_dim - position - 1 # 数组索引 7,6,5,...,0
```

```
        # x 是样本集的记录，来自 a[i]b[i]; y 是样本集对应的标签，来自 c[i]
```

```
        X = np.array([[a[indx],b[indx]]])
```

```
        y = np.array([[c[indx]]]).T
```

```
        # 隐含层 (input ~+ prev_hidden)
```

1. np.dot(X,synapse_I): 从输入层传播到隐含层: 输入层的数据* (输入层—隐含层的权值)

2. np.dot(layer_1_values[-1],synapse_h): 从上一次的隐含层[-1]到当前的隐含层: 上一次的隐含层权值*当前隐含层的权值

```
        # 3. sigmoid(input + prev_hidden)
```

```
        layer_1 = sigmoid(np.dot(X,synapse_I) +np.dot(layer_1_values[-1],synapse_h))
```

```
        # 输出层 (new binary representation)
```

```
        # np.dot(layer_1,synapse_O): 它从隐含层传播到输出层，即输出一个预测值。
```

```
        layer_2 = sigmoid(np.dot(layer_1,synapse_O))
```

```
        # 计算预测误差
```

```
        layer_2_error = y - layer_2
```

```
        layer_2_deltas.append((layer_2_error)*dlogit(layer_2)) # 保留输出层每个
```

时刻的误差，用于反向传播

```

overallError += np.abs(layer_2_error[0]) # 计算二进制的误差绝对值的总和，标量

d[indx] = np.round(layer_2[0][0]) # 存储预测的结果——显示使用

layer_1_values.append(copy.deepcopy(layer_1)) # 存储隐含层的权值，以便在下
次时间迭代中能用

future_layer_1_delta = np.zeros(hidden_dim) # 初始化下一隐含层的误差
# 反向传播：从最后一个时间点开始，反向一直到第一个： position 索引 0,1,2,...,7
for position in xrange(binary_dim):
    X = np.array([[a[position],b[position]]])

    layer_1 = layer_1_values[-position-1] # 从列表中取出当前的隐含层。从最后一层
    开始，-1, -2, -3
    prev_layer_1 = layer_1_values[-position-2] # 从列表中取出当前层的前一隐含层。

    layer_2_delta = layer_2_deltas[-position-1] # 取出当前输出层的误差
    # 计算当前隐含层的误差：
    # future_layer_1_delta.dot(synapse_h.T): 下一隐含层误差*隐含层权重
    # layer_2_delta.dot(synapse_o.T): 当前输出层误差*输出层权重
    # dlogit(layer_1): 当前隐含层的导数
    layer_1_delta = (future_layer_1_delta.dot(synapse_h.T) + layer_2_delta.
    dot(synapse_o.T)) * dlogit(layer_1)

    # 反向更新权重：更新顺序输出层→隐含层→输入层
    # np.atleast_2d: 输入层 reshape 为 2d 的数组
    synapse_o_update += np.atleast_2d(layer_1).T.dot(layer_2_delta)
    synapse_h_update += np.atleast_2d(prev_layer_1).T.dot(layer_1_delta)
    synapse_i_update += X.T.dot(layer_1_delta)

    future_layer_1_delta = layer_1_delta # 下一隐含层的误差
# 更新三个权值
synapse_i += synapse_i_update * alpha
synapse_o += synapse_o_update * alpha
synapse_h += synapse_h_update * alpha
# 所有权值更新项归零
synapse_i_update *= 0;    synapse_o_update *= 0;    synapse_h_update *= 0

# 逐次打印输出
showresult(j, overallError, d, c, a_int, b_int)

```


相信读者通过公式和代码比对，能够理解源码的含义。由于篇幅限制，这里就不给出 Simple-RNN 的案例代码，Simple-RNN 的 Python 实现源码参见网址：<http://www.threadweb.cn/thread-1595-1-1.html>，感兴趣的读者可从上述网址下载。

虽然 Simple-RNN 实现了循环网络的最初雏形，但是在实际应用中，使用得并不多。主要原因有如下两个。

- 因为我们的网络是根据输入而展开的，输入越长，展开的网络就越深，对于“深度”网络训练的困难最常见的是“梯度爆炸 (Gradient Explode)”和“梯度消失 (Gradient Vanish)”的问题。这在 Yoshua Bengio 的论文 *On the difficulty of training recurrent neural networks* 有详细的说明，这里不再说明。
- Simple-RNN 善于基于先前的词来预测下一个词，但在一些更加复杂的场景中，例如，“我出生在法国……我能讲一口流利的法语。”“法国”和“法语”则需要更长时间的预测，而当上下文之间的间隔不断增大时，Simple-RNN 会丧失学习到连接如此远的信息的能力。

9.3.2 LSTM 原理

Long Short Term Memory Networks，即人们常称作“LSTM”的神经网络，它是 RNN 一个变种，专门用于解决 Simple-RNN 的上述两个问题。Hochreiter & Schmidhuber 早在 1997 年就提出了 LSTM 网络，后来 Alex Graves 对其进行了改良和推广。在 NLP 的很多问题上，LSTM 都取得了巨大的成功，并得到了广泛的使用。

LSTM 通过对循环层的刻意设计来避免长期依赖和梯度消失等问题。长期信息的记忆在 LSTM 中是默认行为，而无须付出代价即可获得此能力。

从网络主体上来看，RNN 和 LSTM 是相似的，都具有一种循环神经网络的链式形式。在标准的 RNN 中，这个循环节点只有一个非常简单的结构，如一个 tanh 层。LSTM 的内部要复杂得多，在循环的阶段内部拥有更复杂的结构，即 4 个不同的层来控制信息的交互。

1. LSTM 整体架构图与图例

为了更清晰地观察 LSTM 的内部结构，这里借用了 colah's blog (<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>) 中相关的图片，如图 9.17 所示，并根据 9.3.3 节给出的程序代码，在参数上做了修改。

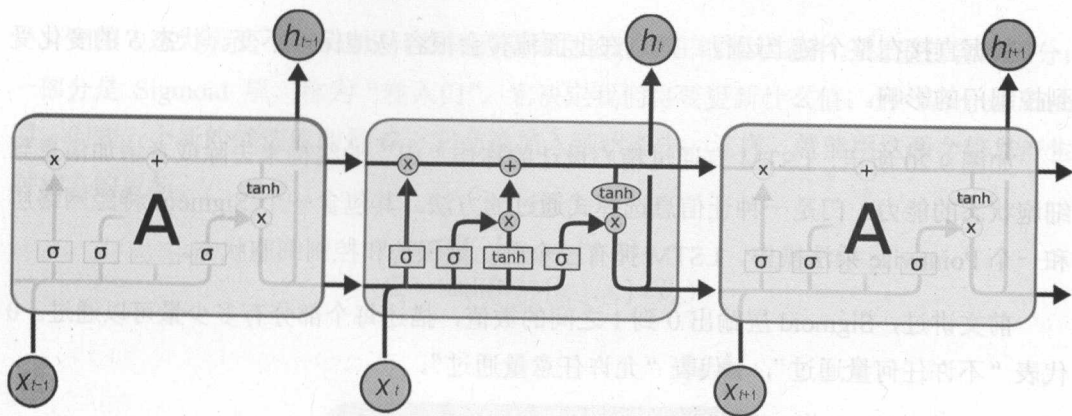


图 9.17 LSTM 中 4 个交互的层

首先给出一些必要的图例（见图 9.18）。

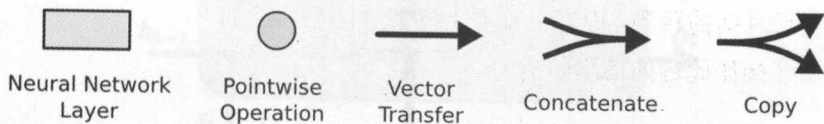


图 9.18 LSTM 神经网络的图例

在图 9.18 中，黄色的矩阵表示学习到的神经网络层；粉色的圆圈代表逐点(Pointwise)的运算，如“+”表示向量求和，“ \times ”表示向量的乘积；每一条黑线传输着一个向量，从一个节点的输出到其他节点的输入，而合在一起的线表示向量的连接；分开的线表示内容被复制，然后分发到不同的位置。

如图 9.19 所示，LSTM 中在图上方贯穿运行的水平线指示了隐含层中神经细胞 (cell) 的状态 S ，细胞状态类似于传送带，只与少量的线交互。

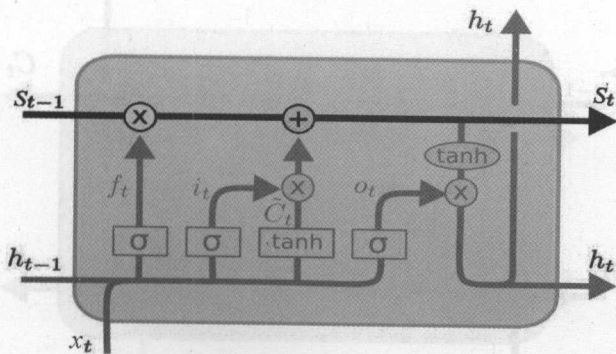


图 9.19 细胞的状态

数据直接在整个链上运行，信息在上面流传会很容易地保持不变。状态 S 的变化受到控制门的影响。

如图 9.20 所示，LSTM 有通过精心设计的称作“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。其包含一个 Sigmoid 神经网络层和一个 Pointwise 乘法操作。LSTM 拥有三个门，来保护和控制细胞状态。

前文讲过，Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 代表“允许任意量通过”。

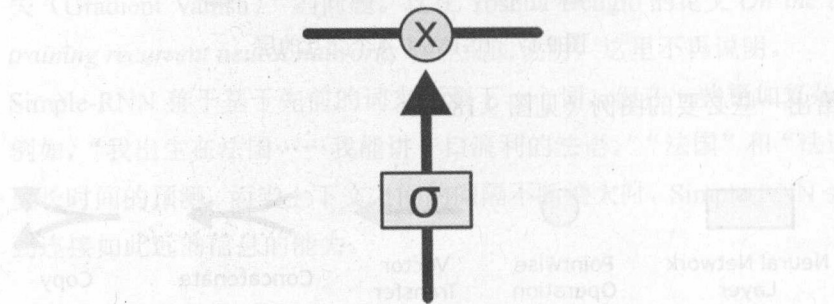


图 9.20 控制门

2. 逐步理解 LSTM

如图 9.21 所示，首先，决定从细胞状态中丢弃什么信息。这个决策是通过一个称为“忘记门”的层来完成的。该门会读取 h_{t-1} 和 x_t ，使用 sigmoid 函数输出一个在 0~1 之间的数值，输出给在状态 S_{t-1} 中每个细胞的数值，1 表示“完全保留”，0 表示“完全舍弃”。

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

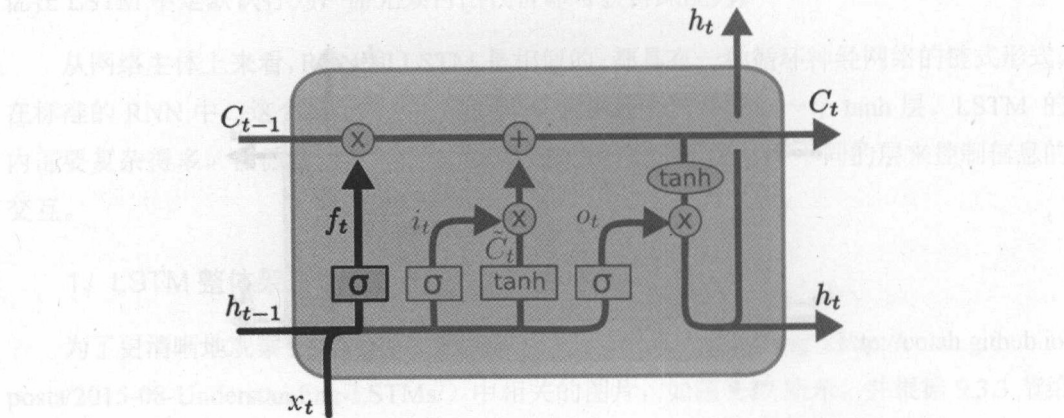


图 9.21 遗忘门

如图 9.22 所示，然后确定什么样的新信息被存放在细胞状态中。这里包含两部分：一部分是 Sigmoid 层，称为“输入门”，它决定我们将要更新什么值；另一部分是 tanh 层，创建一个新的候选值向量 G_t ，它会被加入到状态中。这样，就能用这两个信息产生对状态的更新。

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$G_t = \tanh(W_G \cdot [h_{t-1}, x_t] + b_G)$$

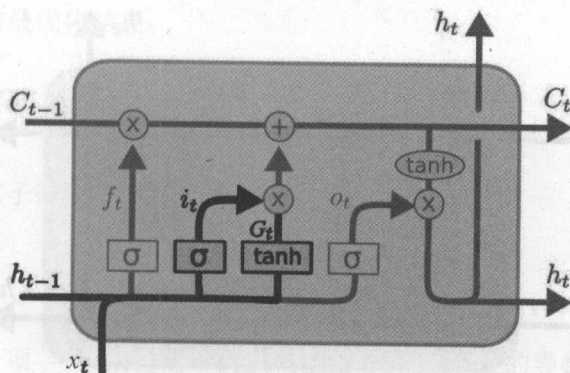


图 9.22 输入与更新

如图 9.23 所示，现在是更新旧细胞状态的时间了， S_{t-1} 更新为 S_t 。前面的步骤已经决定了将会做什么，现在就是实际去完成。把旧状态与 f_t 相乘，丢弃掉我们确定需要丢弃的信息。接着加上 $i_t * G_t$ 。这就是新的候选值，根据更新每个状态的程度进行变化。

在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的类别信息并添加新的信息的地方。

$$S_t = f_t * S_{t-1} + i_t * G_t$$

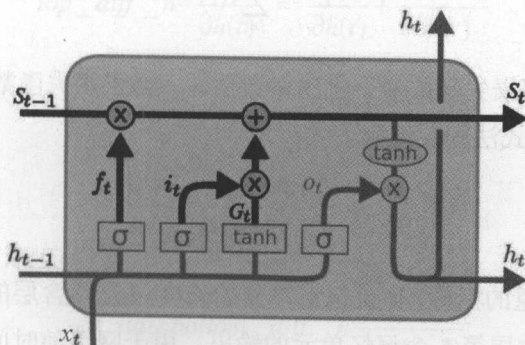


图 9.23 更新细胞状态

如图 9.24 所示, 最终, 需要确定输出什么值。这个输出将会基于细胞状态, 但也是一个过滤后的版本。首先, 运行一个 Sigmoid 层来确定细胞状态的哪个部分将输出出去。接着, 把细胞状态通过 \tanh 进行处理 (得到一个在 $-1 \sim 1$ 之间的值) 并将它和 Sigmoid 门的输出相乘, 最终仅仅会输出我们确定输出的那部分。

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

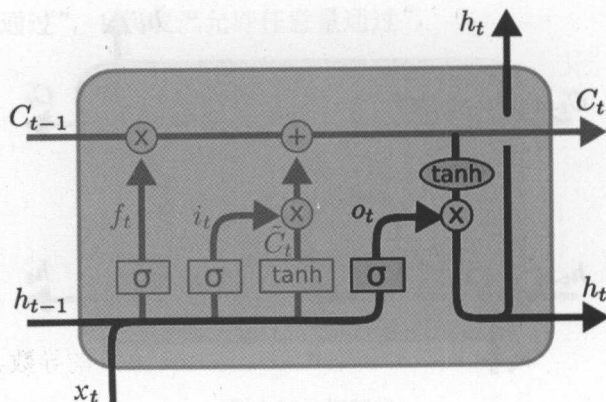


图 9.24 输出信息

与 RNN 相同, 都要最小化损失函数 $l(t)$ 。下面用 $h(t)$ 表示当前时刻的隐含层输出, $y(t)$ 表示当前时刻的输出标签, 参考在后面的代码使用的是平方差损失函数, 则损失函数被表示为:

$$l(t) = f(h(t), y(t)) = \|h(t) - y(t)\|^2$$

全局化的损失函数如下。

$$L = \sum_{t=1}^T l(t)$$

通过梯度法, 实现损失函数最小化的参数估计。由于损失函数 $l(t)$ 依赖于隐含层 $h(t)$ 和输出层 $y(t)$, 根据链式法则, 得到下式:

$$\frac{\partial L}{\partial w} = \sum_{t=1}^T \sum_{i=1}^M \frac{\partial L}{\partial h_i(t)} \cdot \frac{\partial h_i(t)}{\partial w}$$

这里 w 是模型权重的标量, M 是记忆单元的长度, i 是隐含层的第 i 个记忆单元, $h_i(t)$ 是一个标量, 表示隐含层第 i 个记忆单元的输出。由于网络随时间前向传播, 改变 $h_i(t)$ 将不会影响到先于时间 t 的损失。

引入一个变量 $L(t)$, 它表达了第 t 步开始到结束的损失, 如果 $t=1$, 则计算全局误差。

$$L(t) = \sum_{s=t}^T l(s)$$

上述函数变更如下。

$$\frac{\partial L}{\partial w} = \sum_{t=1}^T \sum_{i=1}^M \frac{\partial L(t)}{\partial h_i(t)} \cdot \frac{\partial h_i(t)}{\partial w}$$

求解这个式子的最优化结果:

$$L(t) = \begin{cases} l(t) + L(t+1) & \text{if } t < T \\ l(t) & \text{if } t = T \end{cases}$$

联立上述两个式子得到:

$$\frac{\partial L(t)}{\partial h(t)} = \frac{\partial l(t)}{\partial h(t)} + \frac{\partial L(t+1)}{\partial h(t)}$$

上式的右侧第一项 $\frac{\partial l(t)}{\partial h(t)}$ 来自简单的损失函数 $l(t)$ 的时间 t 的导数。第二项 $\frac{\partial L(t+1)}{\partial h(t)}$ 的

本质是一个循环项, 它表明, 计算当前节点的导数的信息时, 需要下一节点的导数信息。这与 RNN 网络反向传播的过程相同, 这里不再详细说明。

9.3.3 节给出的是 LSTM 的一个案例代码, 读者可以对照上述公式来运行 LSTM 的全过程。在执行之前, 先对 BPTT 的过程进行说明。这个说明有助于读者更清晰地理解源代码的逻辑。

BPTT 算法对应的公式为 $\frac{\partial L(t)}{\partial h(t)} = \frac{\partial l(t)}{\partial h(t)} + \frac{\partial L(t+1)}{\partial h(t)}$ 。变量如下。

$$\begin{aligned} \text{top_diff_h} &= \frac{\partial L(t)}{\partial h(t)} = \frac{\partial l(t)}{\partial h(t)} + \frac{\partial L(t+1)}{\partial h(t)} \\ \text{top_diff_s} &= \frac{\partial L(t+1)}{\partial h(t)} \end{aligned}$$

然后计算:

$$\begin{aligned} \text{self.state.bottom_diff_s} &= \frac{\partial L(t)}{\partial s(t)} \\ \text{self.state.bottom_diff_h} &= \frac{\partial L(t)}{\partial h(t-1)} \end{aligned}$$

这些值将随时间反向传播, 加入求导后的代码如下。

$$self.param.wi_diff = \frac{\partial L}{\partial w_i}$$

$$self.param.bi_diff = \frac{\partial L}{\partial b_i}$$

关于剩下的内容，可参见 9.3.3 节的代码实现。

9.3.3 LSTM 的 Python 实现

下面给出整个 LSTM 实现的 Python 版本。该版本仅用于学习，便于读者了解算法执行的每个细节。

1. LSTM 类

```
# -*- coding: UTF-8 -*-
import os, sys
import numpy as np
import random
import math

# 网络激活函数
def sigmoid(x):
    return 1. / (1 + np.exp(-x))

# 创建一个范围在[a,b)的维度为args的随机矩阵
def rand_arr(a, b, *args):
    np.random.seed(0)
    return np.random.rand(*args) * (b - a) + a

# 损失函数类：
class LossLayer:
    @classmethod # 计算平方损失
    def loss(self, pred, label):
        return (pred[0] - label) ** 2
    @classmethod # 实际输出和期望输出之间的差
    def bottom_diff(self, pred, label):
        diff = np.zeros_like(pred)
        diff[0] = 2 * (pred[0] - label)
        return diff

# lstm 参数类
class LstmParam:
    def __init__(self, mem_cell_ct, x_dim):
        self.mem_cell_ct = mem_cell_ct
        self.x_dim = x_dim
```

```

concat_len = x_dim + mem_cell_ct
# 初始化各个门的权重矩阵
self.wg = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
self.wi = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
self.wf = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
self.wo = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
# 初始化各个门的偏置矩阵
self.bg = rand_arr(-0.1, 0.1, mem_cell_ct)
self.bi = rand_arr(-0.1, 0.1, mem_cell_ct)
self.bf = rand_arr(-0.1, 0.1, mem_cell_ct)
self.bo = rand_arr(-0.1, 0.1, mem_cell_ct)
# 损失函数的导数: 权重、偏置
self.wg_diff = np.zeros((mem_cell_ct, concat_len))
self.wi_diff = np.zeros((mem_cell_ct, concat_len))
self.wf_diff = np.zeros((mem_cell_ct, concat_len))
self.wo_diff = np.zeros((mem_cell_ct, concat_len))
self.bg_diff = np.zeros(mem_cell_ct)
self.bi_diff = np.zeros(mem_cell_ct)
self.bf_diff = np.zeros(mem_cell_ct)
self.bo_diff = np.zeros(mem_cell_ct)
# 更新参数
def apply_diff(self, lr = 1):
    self.wg -= lr * self.wg_diff
    self.wi -= lr * self.wi_diff
    self.wf -= lr * self.wf_diff
    self.wo -= lr * self.wo_diff
    self.bg -= lr * self.bg_diff
    self.bi -= lr * self.bi_diff
    self.bf -= lr * self.bf_diff
    self.bo -= lr * self.bo_diff
# 重置权重和偏置的差异为 0
self.wg_diff = np.zeros_like(self.wg)
self.wi_diff = np.zeros_like(self.wi)
self.wf_diff = np.zeros_like(self.wf)
self.wo_diff = np.zeros_like(self.wo)
self.bg_diff = np.zeros_like(self.bg)
self.bi_diff = np.zeros_like(self.bi)
self.bf_diff = np.zeros_like(self.bf)
self.bo_diff = np.zeros_like(self.bo)

# lstm 状态类
class LstmState:
    def __init__(self, mem_cell_ct, x_dim):

```

```

self.g = np.zeros(mem_cell_ct) # 候选值向量
self.i = np.zeros(mem_cell_ct) # 输入门
self.f = np.zeros(mem_cell_ct) # 忘记门
self.o = np.zeros(mem_cell_ct) # 输出门
self.s = np.zeros(mem_cell_ct) # 内部状态
self.h = np.zeros(mem_cell_ct) # 实际输出
self.bottom_diff_h = np.zeros_like(self.h)
self.bottom_diff_s = np.zeros_like(self.s)
self.bottom_diff_x = np.zeros(x_dim)

class LstmNode:
    def __init__(self, lstm_param, lstm_state):
        self.state = lstm_state # store reference to parameters and to activations
        self.param = lstm_param
        self.x = None # 输入层（非循环层）节点
        self.xc = None # 非循环(non-recurrent)层输入+ 循环(recurrent)层输入
        #
    def bottom_data_is(self, x, s_prev = None, h_prev = None):
        # if this is the first lstm node in the network
        if s_prev is None: s_prev = np.zeros_like(self.state.s)
        if h_prev is None: h_prev = np.zeros_like(self.state.h)
        # 存储数据用于反向传播
        self.s_prev = s_prev
        self.h_prev = h_prev

        xc = np.hstack((x, h_prev)) # xc(t)=[x(t),h(t-1)]
        self.state.g = np.tanh(np.dot(self.param.wg, xc) + self.param.bg) # 候
        # 选值向量
        self.state.i = sigmoid(np.dot(self.param.wi, xc) + self.param.bi) # 输入门
        self.state.f = sigmoid(np.dot(self.param.wf, xc) + self.param.bf) # 忘记门
        self.state.o = sigmoid(np.dot(self.param.wo, xc) + self.param.bo) # 输出门
        self.state.s = self.state.g * self.state.i + s_prev * self.state.f # 内部状态
        self.state.h = self.state.s * self.state.o # 实际输出
        self.x = x
        self.xc = xc

        # 注意 top_diff_s 沿固定误差传递
    def top_diff_is(self, top_diff_h, top_diff_s):
        ds = self.state.o * top_diff_h + top_diff_s
        do = self.state.s * top_diff_h
        di = self.state.g * ds
        dg = self.state.i * ds

```



```

df = self.s_prev * ds

# diffs w.r.t. vector inside sigma / tanh function
di_input = (1. - self.state.i) * self.state.i * di
df_input = (1. - self.state.f) * self.state.f * df
do_input = (1. - self.state.o) * self.state.o * do
dg_input = (1. - self.state.g ** 2) * dg

# 输入层的误差
self.param.wi_diff += np.outer(di_input, self.xc)
self.param.wf_diff += np.outer(df_input, self.xc)
self.param.wo_diff += np.outer(do_input, self.xc)
self.param.wg_diff += np.outer(dg_input, self.xc)
self.param.bi_diff += di_input
self.param.bf_diff += df_input
self.param.bo_diff += do_input
self.param.bg_diff += dg_input

# 计算底层 (bottom) 误差
dxc = np.zeros_like(self.xc)
dxc += np.dot(self.param.wi.T, di_input)
dxc += np.dot(self.param.wf.T, df_input)
dxc += np.dot(self.param.wo.T, do_input)
dxc += np.dot(self.param.wg.T, dg_input)

# 存储底层 (bottom) 误差
self.state.bottom_diff_s = ds * self.state.f
self.state.bottom_diff_x = dxc[:self.param.x_dim]
self.state.bottom_diff_h = dxc[self.param.x_dim:]

class LstmNetwork():
    def __init__(self, lstm_param):
        self.lstm_param = lstm_param
        self.lstm_node_list = [] # 状态序列
        self.x_list = [] # 输入序列

    def y_list_is(self, y_list, loss_layer): # 使用损失更新预测标签
        assert len(y_list) == len(self.x_list)
        idx = len(self.x_list) - 1
        # 计算损失: 第一个节点仅得到来自目标标签的误差
        loss = loss_layer.loss(self.lstm_node_list[idx].state.h, y_list[idx])
        diff_h = loss_layer.bottom_diff(self.lstm_node_list[idx].state.h, y_list[idx])

```

```

        # here s is not affecting loss due to h(t+1), hence we set equal to zero
        diff_s = np.zeros(self.lstm_param.mem_cell_ct)
        self.lstm_node_list[idx].top_diff_is(diff_h, diff_s)
        idx -= 1

    while idx >= 0:
        loss += loss_layer.loss(self.lstm_node_list[idx].state.h, y_list[idx])
# 累计误差

        # 计算各种门的误差
        diff_h = loss_layer.bottom_diff(self.lstm_node_list[idx].state.h,
y_list[idx]) #
        diff_h += self.lstm_node_list[idx + 1].state.bottom_diff_h #
        diff_s = self.lstm_node_list[idx + 1].state.bottom_diff_s #
        self.lstm_node_list[idx].top_diff_is(diff_h, diff_s) #
        idx -= 1

    return loss

# 清空输入层
def x_list_clear(self):
    self.x_list = []

def x_list_add(self, x): #创建输入序列 x 和初始状态节点
    self.x_list.append(x)
    if len(self.x_list) > len(self.lstm_node_list):
        lstm_state = LstmState(self.lstm_param.mem_cell_ct, self.lstm_param.
x_dim) # 初始状态节点
        self.lstm_node_list.append(LstmNode(self.lstm_param, lstm_state))

# 获取 t-1 个输入序列
idx = len(self.x_list) - 1
if idx == 0:
    self.lstm_node_list[idx].bottom_data_is(x)
else: # 创建 lstm 各种门所需的 t-1 状态
    s_prev = self.lstm_node_list[idx - 1].state.s
    h_prev = self.lstm_node_list[idx - 1].state.h
    self.lstm_node_list[idx].bottom_data_is(x, s_prev, h_prev)

```

2. 执行函数

```

# -*- coding: UTF-8 -*-
import os, sys
import numpy as np
from lstm import *

```

```

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')

np.random.seed(0)

def gensamples(x_dim):
    ylabels = [-0.5, 0.2, 0.1, -0.5]
    xinputs = [np.random.random(x_dim) for i in ylabels] # 对应输出矩阵的一系列随机数
    return xinputs, ylabels

if __name__ == "__main__":
    x_dim = 50 # 输出维度
    maxiter = 100 # 最大迭代次数

    # input_val->X: 50 维的随机数向量; y_list->y: 每个  $X_i$  向量对应的一个  $y$  的输出值,
    #  $X_i[0:50] \rightarrow y_i$ 
    input_val_arr, y_list = gensamples(x_dim)

    # 初始化 lstm 各部分参数
    mem_cell_ct = 100 # 存储单元维度
    concat_len = x_dim + mem_cell_ct # 输入维度与存储单元维度之和

    lstm_param = LstmParam(mem_cell_ct, x_dim) # 初始化 lstm 神经网络的参数
    lstm_net = LstmNetwork(lstm_param) # 创建 lstm 神经网络对象
    #主程序:
    for cur_iter in xrange(maxiter):
        ypredlist = []
        for ind in xrange(len(y_list)):
            lstm_net.x_list_add(input_val_arr[ind])
            ypredlist.append((ind, lstm_net.lstm_node_list[ind].state.h[0]))

        loss = lstm_net.y_list_is(y_list, LossLayer) # 计算全局损失
        lstm_param.apply_diff(lr=0.1) #
        lstm_net.x_list_clear() #

        if (cur_iter+1)%10==0: # 输出迭代各个阶段的值
            print "cur iter: ", cur_iter
            print "y_pred: ", ypredlist
            print "loss: ", loss

```

一个四列

3. 输出结果

```
cur iter: 9
y_pred: [(0, -0.30918477256783872), (1, -0.14732219806187546), (2, -0.15495345472929759),
(3, -0.29029294481375578)]
loss: 0.26602147336
cur iter: 19
y_pred: [(0, -0.38409063297396573), (1, -0.10630445803949244), (2, -0.12412162138802935),
(3, -0.37845034415198858)]
loss: 0.17226222239
cur iter: 29
y_pred: [(0, -0.41986418632264572), (1, -0.077439243435045352), (2, -0.097247992104092351),
(3, -0.42475339468242951)]
loss: 0.127963104432
cur iter: 39
y_pred: [(0, -0.44039365243178058), (1, -0.05940774053788752), (2, -0.078468153734407617),
(3, -0.44766487445042785)]
loss: 0.105435139785
cur iter: 49
y_pred: [(0, -0.45370090557486359), (1, -0.047704273922078329), (2, -0.065475773582741523),
(3, -0.46014229444177157)]
loss: 0.092471881799
cur iter: 59
y_pred: [(0, -0.46284093342326327), (1, -0.039649304074017955), (2, -0.056131579632202749),
(3, -0.4677425990600011)]
loss: 0.0842301952459
cur iter: 69
y_pred: [(0, -0.46934208481704998), (1, -0.033817602589949217), (2, -0.049131574632131657),
(3, -0.47283057482851498)]
loss: 0.0785889832607
cur iter: 79
y_pred: [(0, -0.47410142969798369), (1, -0.029421553566928434), (2, -0.043704000037573132),
(3, -0.47648821545729736)]
loss: 0.0745086288239
cur iter: 89
y_pred: [(0, -0.47768068403126046), (1, -0.02599952437834465), (2, -0.039376054576753423),
(3, -0.47925662987435835)]
loss: 0.0714299088781
cur iter: 99
y_pred: [(0, -0.48044164497776687), (1, -0.0232657206358283), (2, -0.035845123130771074),
(3, -0.4814314917266011)]
loss: 0.0690287982401
```

9.4 深度学习框架与应用

目前,越来越多的深度学习开源框架在网络上发布出来。下面给出如下几个有代表性的框架帮助读者学习。

- ❑ Caffe。该项目源自加州伯克利分校的贾扬清, Caffe 由 C++写成, 由于其高效的运行效率, 发布之后很快被广泛应用, 包括 Pinterest 这样的 Web 大户。目前 Caffe 主要用于卷积神经网络的学习, 在机器视觉领域仍旧是应用最广泛的深度学习框架。
- ❑ Torch。Torch 诞生已经有十年之久, 是易于使用且高效的计算框架, 由于 Facebook 开源了大量 Torch 的深度学习模块和扩展, 使 Torch 很快广泛地发展起来, 成为受欢迎的深度学习计算框架之一。另外一个特殊之处是采用了不怎么流行的编程语言 Lua。
- ❑ Tensorflow。2015 年 11 月 9 日, Google 官方称, Google Research 宣布开源了第二代机器学习系统 TensorFlow, 它也是用 C++语言开发的, 前端脚本使用 Python, 并针对先前的 DistBelief 的短板有了各方面的加强。与前面的框架不同, TensorFlow 不仅实现了传统的深度学习算法, 而且不断提出和发布新的、有价值的算法。现在, Tensorflow 越来越成为深度学习发展的前沿。

9.4.1 Keras 框架介绍

1. Keras 介绍

Keras(<https://keras.io/>)是一个基于 Theano 或 TensorFlow 作为后端的深度学习框架, 它的设计参考了 Torch, 用 Python 语言编写, 是一个高度模块化的神经网络库。因此, 在结构上是极度简化、便于设计的深度学习第三方库。基于 Python 整合多种后端开发, 充分发挥了 GPU 和 CPU 运算能力。其开发目的是利用快速原型的方法设计基于深度学习的各种实验。适合前期的网络原型设计、支持卷积网络和递归网络及两者整合的结果、支持任意的连接范式、能够在 GPU 和 CPU 上无缝连接运行。

Keras 的源代码可以从 <https://github.com/fchollet/keras> 下载。源代码结构简单, 便于阅读, 感兴趣的读者可以从源码学习各种深度学习网络的实现细节。

指导原则如下。

- ❑ **模块化。**模型被理解为一个序列或独立的图等完全可配置的模块，并使用尽可能少的限制被插到一起。特别是神经网络层、代价函数、优化方法、初始化方案，激活函数，正规化的方案都是独立的模块，它们可以结合起来构造新的模型。
- ❑ **极简主义。**每个模块的设置尽量简短。每一段代码在第一次读应该是透明的。没有黑魔法，不会伤害迭代速度和创新能力。
- ❑ **易扩展性。**新的模块添加起来极其简单（如新类和函数），并且现有模块已经提供了充足的例子。能够轻松地创建新的模块提升总体性能，使得 Keras 适用于最先进的研究任务。
- ❑ **基于 Python 的编码。**没有以声明形式提供的独立的模型配置文件。模型使用 Python 代码来描述，其结构紧凑、易于调试，并允许扩展。

2. Keras 的安装与结构

由于 Keras 不是独立的深度学习 (DL) 算法包，其安装必须依赖于后端系统的安装，特别是对 GPU 的支持，因此其安装并非那么简单。在 Windows 系统下，需要事先安装 Theano 开发包，并使 Theano 支持 GPU 运算；在 Linux 系统下，可以选择 Theano 和 TensorFlow 两种后端，当然每种后端都要支持 GPU 运算。

本章重点讲解 Windows 下的 Keras 安装。关于 Theano 的安装建议读者参考 Theano 官方网站的安装教程，网址是 <http://deeplearning.net/software/theano/install.html>。网页内包含 Windows 下的安装介绍，更多的细节建议参考《机器学习算法原理与编程实践》中第 10 章中的更多细节。

这里需要说明的是，目前 CUDA 7.5 最高仅支持 Visual Studio 2013，如果你使用的是更高的版本，必须降级 (Downgrade) 到 Visual Studio 2013 或更低的版本上。

Keras 提供了一套完整的深度学习网络的实现。该实现主要包括一套完整的参数和函数 API。Keras 提供了一套完整的教程文档，从文档我们了解到 Keras 主要包含如下几部分内容。

Models (模型)。这是 Keras 最主要的模块。目前比较流行的模型是序列化模型，文档内对序列化模型给出了讲解，用户都是从操作模型来实现网络架构的设计的，但在操作这一层之前，需要对网络的各个细节有充分的了解。后面定义了各种基本组件，都通过模型层将它们组合起来。

Layers (层)。模块是由不同类型的神经网络层构成的，Layers 模块包含 Core、

Convolutional、Recurrent、Advanced_activations、Normalization、Embeddings 这几种。对于 NLP 而言，Core 和 Recurrent 循环层是我们关注的重点。在 Core 中包含如下一系列层的种类。

- ❑ Flatten 层。CNN 的全连接层之前需要把二维特征图 Flatten 成为一维的。
- ❑ Reshape。CNN 输入时将一维的向量变成二维的，相当于 Flatten 的逆操作。
- ❑ Dense。就是网络中的隐含层（英文 Dense 是稠密的意思）。
- ❑ Convolutional。Convolutional 层基本就是 Theano 的 Convolution2D 的封装。

Optimizers（优化算法）。Optimizers 包含了一些优化的方法，比如最基本的随机梯度下降 SGD。另外，还有 Adagrad、Adadelta、RMSprop、Adam，一些新的方法以后也会被不断添加进来。

Objectives（目标函数）。这是神经网络的目标函数模块，Keras 提供了 Mean_squared_error、Mean_absolute_error、Squared_hinge、Hinge、Binary_crossentropy、Categorical_crossentropy 这几种目标函数。

Activations（激活函数）。Keras 提供了 Linear、Sigmoid、Hard_sigmoid、Tanh、Softplus、Relu、Softplus，另外，Softmax 也放在 Activations 模块中（笔者觉得放在 Layers 模块中更合理些）。此外，像 LeakyReLU 和 PReLU 这种比较新的激活函数，Keras 也提供。

Initializations（初始化方法）。这是估计参数初始化模块，在添加 Layer 的时候调用 Init 进行初始化。Keras 提供了 Uniform、Lecun_uniform、Normal、Orthogonal、Zero、Glorot_normal、He_normal 这几种。

Preprocessing（预处理）。这是预处理模块，包括序列数据的处理、文本数据的处理和图像数据的处理等。

有关上述模块的细节，后面会提供一个实例进行讲解。

3. 使用 Keras 设计网络架构

使用 Keras 框架来实现深度学习的快速原型，原因有很多。其中一个主要原因是 Keras 在教程文档中提供了多种网络框架以供用户选择，用户可以以现有的框架为基础，根据自己的专门用途设计出新的网络原型。而构建各种网络原形的代码量很少、设计速度很快，便于将专有的网络框架快速应用于自身 NLP 任务实验中。两种常用的网络架构如下。

（1）堆叠的 LSTM 网络架构。

有状态循环模型（LSTM）将前一批样本生成的内部状态（记忆）保存，在下一批

重用。它能处理更长的序列，并降低计算复杂性。

在图 9.25 所示的模型中，堆叠了 3 个 LSTM 层，使模型有能力进行更高层的时序表达。前两个 LSTMs 返回了全部的输出序列，但最后一个仅仅返回了其最后一步的输出序列。这样，丢弃了时间的维度（这便将输入序列转换为一个独立的向量）。

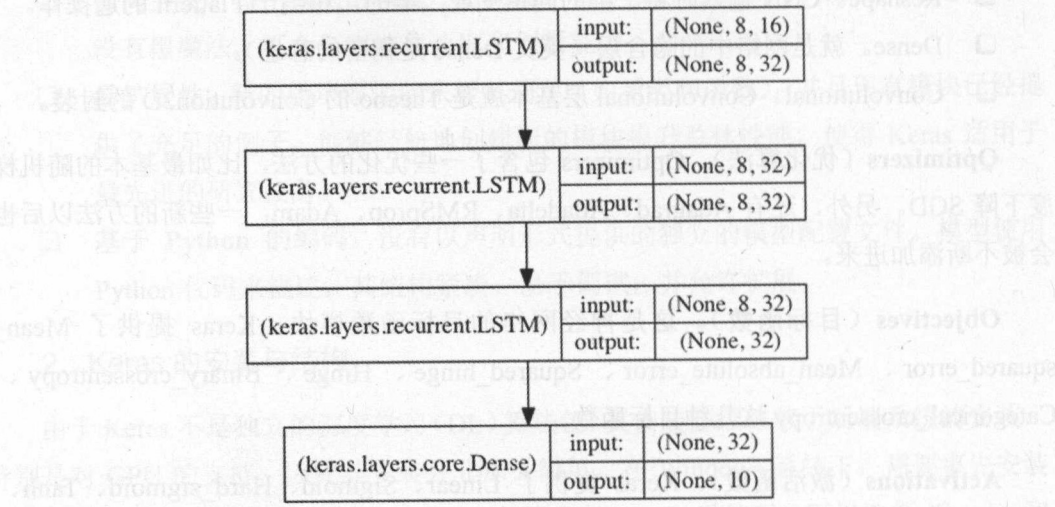


图 9.25 堆叠的神经网络架构

(2) LSTM 层的合并网络架构。

如图 9.26 所示，在该模型中，两个输入序列由两个独立的 LSTM 模块被编码成向量。然后这两个向量串联起来，一个全连接的网络位于串联表达的顶层。

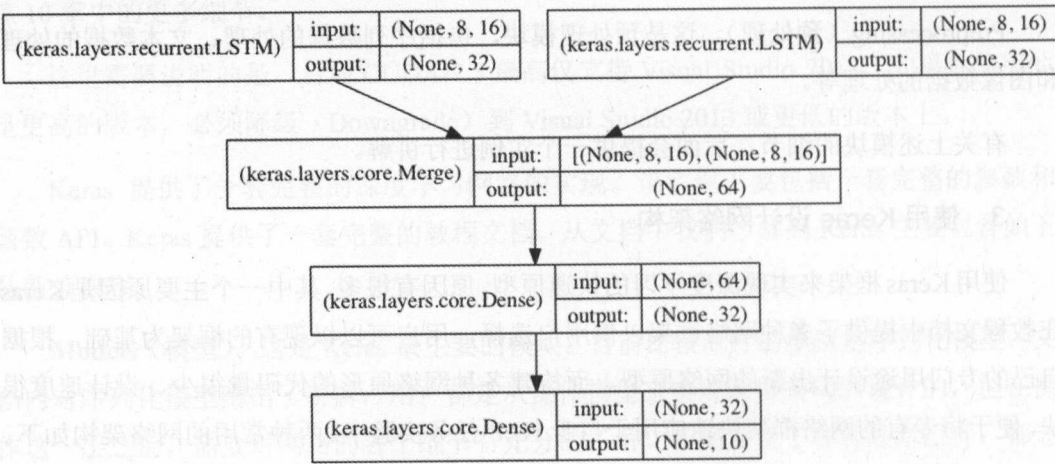


图 9.26 合并网络架构

9.4.2 Keras 序列标注

下面以一个中文分词的例子来展示 Keras 序列标注的过程。

1. 必须导入模块包和语料库文件

```
# -*- coding: utf-8 -*-
import os, sys
import numpy as np
from numpy import *
import nltk
import codecs
import pandas as pd
from nltk.probability import FreqDist
from gensim.models import word2vec
from cPickle import load, dump
from keras.preprocessing import sequence
from keras.optimizers import SGD, RMSprop, Adagrad
from keras.utils import np_utils
from keras.models import Sequential, Graph
from keras.layers.core import Dense, Dropout, Activation, TimeDistributedDense
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM, GRU, SimpleRNN
from keras.layers.core import Reshape, Flatten, Dropout
from keras.regularizers import l1, l2
from keras.layers.convolutional import Convolution2D, MaxPooling2D, MaxPooling1D
from sklearn.cross_validation import train_test_split
```

上述为全部需要导入的包，将这些包放到一个称为 `seqlib.py` 的文件中，执行文件在执行过程中根据需要自行引入。受篇幅所限，这里对执行程序的需要包不另作说明，均加以忽略。

下文中的必要函数都和神经网络模型类位于 `seqlib.py` 中。

训练语料库使用的是在第 7 章中介绍过的微软研究院的中文分词语料库：`msr.utf8.txt`。

2. 使用分词语料库生成词向量

(1) 必要的函数：`seqlib.py`。

```
def load_file(input_file): # 读单个文本
    input_data = codecs.open(input_file, 'r', 'utf-8')
    input_text = input_data.read()
```



```

    return input_text
# 我们使用了 gensim 的 word2vec 库
def trainW2V(corpus, epochs=20, num_features = 100, sg=1, \ # word2vec 建模
             min_word_count = 1, num_workers = 4, \
             context = 4, sample = 1e-5, negative = 5):
    w2v = word2vec.Word2Vec(workers = num_workers, sample = sample, size =
num_features, min_count=min_word_count, window = context)
    np.random.shuffle(corpus)
    w2v.build_vocab(corpus)
    for epoch in range(epochs):
        print('epoch' + str(epoch))
        np.random.shuffle(corpus)
        w2v.train(corpus)
        w2v.alpha *= 0.9
        w2v.min_alpha = w2v.alpha
    print("word2vec DONE.")
    return w2v

```

(2) 执行程序: corpus2vector.py。

```

# -*- coding: utf-8 -*-
from seqlib import *
reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')

corpuspath = "msr.utf8.txt"
input_text = load_file(corpuspath)

# word2vec 是一个二维数组
txtwv = [line.split() for line in input_text.split('\n') if line != '']

# word2vec
w2v = trainW2V(txtwv)
w2v.save("wordvector.bin")

```

生成词向量二进制文件: wordvector.bin。

3. 语料预处理

(1) 必要的函数: seqlib.py。

```

def freq_func(input_txt): # nltk 输入文本, 输出词频表
    corpus = nltk.Text(input_txt)
    fdist = FreqDist(corpus)

```

```

w = fdist.keys()
v = fdist.values()
freqdf = pd.DataFrame({'word':w, 'freq':v})
freqdf.sort('freq',ascending =False, inplace=True)
freqdf['idx'] = np.arange(len(v))
return freqdf

def initweightlist(w2v, idx2word, word2idx):    # 初始化权重
    init_weight_wv = []
    for i in range(len(idx2word)):
        init_weight_wv.append(w2v[idx2word[i]])
    # 定义'U'为未登录新字, 'P'为两头padding用途, 并增加两个相应的向量表示
    char_num = len(init_weight_wv)
    idx2word[char_num] = u'U'
    word2idx[u'U'] = char_num
    idx2word[char_num+1] = u'P'
    word2idx[u'P'] = char_num+1
    init_weight_wv.append(np.random.randn(100,))
    init_weight_wv.append(np.zeros(100,))
    return init_weight_wv ,idx2word,word2idx

def character_tagging(input_file, output_file): # 加入标注标签: SBME
    input_data = codecs.open(input_file, 'r', 'utf-8')
    output_data = codecs.open(output_file, 'w', 'utf-8')
    for line in input_data.readlines():
        word_list = line.strip().split()
        for word in word_list:
            if len(word) == 1:
                output_data.write(word + "/S ")
            else:
                output_data.write(word[0] + "/B ")
                for w in word[1:len(word)-1]:
                    output_data.write(w + "/M ")
                output_data.write(word[len(word)-1] + "/E ")
        output_data.write("\n")
    input_data.close()
    output_data.close()

def featContext(sentence, word2idx = '', context = 7):
    predict_word_num = []
    for w in sentence:
        # 文本中的字如果在词典中则转为数字, 如果不在则设置为'U'
        if w in word2idx:

```

```

        predict_word_num.append(word2idx[w])
    else:
        predict_word_num.append(word2idx[u'U'])
    num = len(predict_word_num) # 首尾padding
    pad = int((context-1)*0.5)
    for i in range(pad):
        predict_word_num.insert(0, word2idx[u'P'])
        predict_word_num.append(word2idx[u'P'])
    train_x = []
    for i in range(num):
        train_x.append(predict_word_num[i:i+context])
    return train_x

```

(2) 执行程序: preprocess.py。

```

from seqlib import *
reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')

corpuspath = "msr.utf8.txt"
input_text = load_file(corpuspath)

# 计算词频
txtnltk = [w for w in input_text.split()] # 为计算词频准备的文本格式
freqdf = freq_func(txtnltk) # 计算词频表

# 建立两个映射词典
word2idx = dict((c, i) for c, i in zip(freqdf.word, freqdf.idx))
idx2word = dict((i, c) for c, i in zip(freqdf.word, freqdf.idx))
w2v = word2vec.Word2Vec.load("wordvector.bin")

# 初始化向量
init_weight_wv, idx2word, word2idx = initweightlist(w2v, idx2word, word2idx)
dump(word2idx, open('word2idx.pickle', 'wb'))
dump(idx2word, open('idx2word.pickle', 'wb'))

dump(init_weight_wv, open('init_weight_wv.pickle', 'wb'))

# 读取数据, 将格式进行转换为带 4 种标签 S B M E
output_file = 'msr.tagging.utf8'
character_tagging(corpuspath, output_file)

# 分离 word 和 label

```



```

with open(output_file) as f:
    lines = f.readlines()
    train_line = [[w[0] for w in line.decode('utf-8').split()] for line in lines]
    train_label = [w[2] for line in lines for w in line.decode('utf-8').split()]

# 将所有训练文本转成数字 list
train_word_num = []
for line in train_line:
    train_word_num.extend(featsContext(line, word2idx))

# 持久化
dump(train_word_num, open('train_word_num.pickle', 'wb'))
dump(train_label, open('train_label.pickle', 'wb'))

```

执行结果如下。

词与索引映射文件: word2idx.pickle、idx2word.pickle。

初始化权重向量: init_weight_wv.pickle。

使用 BMES 标签标注后的分词语料: msr.tagging.utf8。

使用 Word2Vec 转换后的词向量和索引训练文件: train_word_num.pickle、train_label.pickle

4. 训练语料

(1) 必要的类: seqlib.py。

```

class Lstm_Net(object): # lstm 神经网络
    def __init__(self):
        self.init_weight = []
        self.batch_size = 128
        self.word_dim = 100
        self.maxlen = 7
        self.hidden_units = 100
        self.nb_classes = 0

    def buildnet(self):
        self.maxfeatures = self.init_weight[0].shape[0] # 词典大小
        self.model = Sequential()
        print 'stacking LSTM...' # 使用了堆叠的 LSTM 架构
        self.model.add(Embedding(self.maxfeatures, self.word_dim, input_length=
self.maxlen))

```

```

self.model.add(LSTM(output_dim=self.hidden_units, return_sequences=True))
self.model.add(LSTM(output_dim=self.hidden_units, return_sequences=False))
self.model.add(Dropout(0.5))
self.model.add(Dense(self.nb_classes))
self.model.add(Activation('softmax'))
self.model.compile(loss='categorical_crossentropy', optimizer='adam')

def train(self,modelname):
    result = self.model.fit(self.train_X, self.Y_train, batch_size=self.
batch_size, nb_epoch=20, validation_data = (self.test_X,self.Y_test), show_accuracy=
True)

    self.model.save_weights(modelname)

def splitset(self,train_word_num,train_label, train_size=0.9, random_state=1):
    self.train_X, self.test_X, train_y, test_y = train_test_split(train_
word_num,train_label, train_size=0.9, random_state=1)

    self.Y_train = np_utils.to_categorical(train_y, self.nb_classes)
    self.Y_test = np_utils.to_categorical(test_y, self.nb_classes)

def predict_num(self,input_num,input_txt, label_dict='', num_dict=''):
#根据输入得到标注推断

    input_num = np.array(input_num)
    predict_prob = self.model.predict_proba(input_num, verbose=False)
    predict_label = self.model.predict_classes(input_num, verbose=False)
    for i , label in enumerate(predict_label[:-1]):
        if i==0: # 如果是首字, 不可为E, M
            predict_prob[i, label_dict[u'E']] = 0
            predict_prob[i, label_dict[u'M']] = 0
        if label == label_dict[u'B']: # 前字为B, 后字不可为B,S
            predict_prob[i+1,label_dict[u'B']] = 0
            predict_prob[i+1,label_dict[u'S']] = 0
        if label == label_dict[u'E']: # 前字为E, 后字不可为M,E
            predict_prob[i+1,label_dict[u'M']] = 0
            predict_prob[i+1,label_dict[u'E']] = 0
        if label == label_dict[u'M']: # 前字为M, 后字不可为B,S
            predict_prob[i+1,label_dict[u'B']] = 0
            predict_prob[i+1,label_dict[u'S']] = 0
        if label == label_dict[u'S']: # 前字为S, 后字不可为M,E
            predict_prob[i+1,label_dict[u'M']] = 0
            predict_prob[i+1,label_dict[u'E']] = 0
        predict_label[i+1] = predict_prob[i+1].argmax()
    predict_label_new = [num_dict[x] for x in predict_label]
    result = [w+'/' +l for w, l in zip(input_txt,predict_label_new)]

```

```

        return ' '.join(result) + '\n'

    def getweights(self, wfname):
        return self.model.load_weights(wfname)

```

(2) 执行训练: segment_lstm.py。

```

from seqlib import *
reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')

train_word_num = load(open('train_word_num.pickle', 'rb'))
train_label = load(open('train_label.pickle', 'rb'))
nb_classes = len(np.unique(train_label))
# 初始字向量格式准备
init_weight_wv = load(open('init_weight_wv.pickle', 'rb'))

# 建立两个词典
label_dict = dict(zip(np.unique(train_label), range(4)))
num_dict = {n:l for l,n in label_dict.iteritems()}

# 将目标变量转为数字
train_label = [label_dict[y] for y in train_label]
# print shape(train_label)
train_word_num = np.array(train_word_num)
# print shape(train_word_num)

# stacking LSTM
modelname = 'my_model_weights.h5'
net = Lstm_Net()
net.init_weight = [np.array(init_weight_wv)]
net.nb_classes = nb_classes
net.splitset(train_word_num, train_label)
print "Train..."
net.buildnet()
net.train(modelname)

```

训练结果如下。

```
my_model_weights.h5
```

5. 执行分词

```
from seqlib import *
```



```

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')

word2idx = load(open('word2idx.pickle','rb'))
train_word_num = load(open('train_word_num.pickle','rb'))
train_label = load(open('train_label.pickle','rb'))
nb_classes = len(np.unique(train_label))
init_weight_wv = load(open('init_weight_wv.pickle', 'rb'))

# 建立两个词典
label_dict = dict(zip(np.unique(train_label), range(4)))
num_dict = {n:l for l,n in label_dict.iteritems()}

temp_txt = u'罗马尼亚的首都是布加勒斯特。'
temp_txt = list(temp_txt)
temp_num = featContext(temp_txt,word2idx = word2idx)

net = Lstm_Net()
net.init_weight = [np.array(init_weight_wv)]
net.nb_classes = nb_classes
net.buildnet()
net.getweights('my_model_weights.h5')
temp = net.predict_num(temp_num,temp_txt,label_dict=label_dict,num_dict=num_dict)
print(temp)

```

执行结果如下。

```

罗/B 马/M 尼/M 亚/E 的/S 首/B 都/E 是/S 布/B 加/M 勒/M 斯/M 特/E 。/S

```

执行结果为标注后的文本，文本的标注规则符合 BMES 规则，分词后的结果如下。

```

罗马尼亚|的|首都|是|布加勒斯特|。

```

9.4.3 依存句法的算法原理

第 6 章讨论了两种主要的句法解析方法：一种是基于乔姆斯基的短语结构句法，另一种就是依存句法，但仅详细讲解了短语结构句法的算法原理。由于最新的依存句法分析用到了深度学习的思想，因此，本节对依存句法的算法原理做出详细的分析。

本节使用基于深度学习的神经网络解析器来学习这个模型。首先给出神经网络模型及其构成组件，9.4.4 节给出语料处理和训练过程的细节。

1. 深度学习模型

图 9.27 展示了基于深度学习的依存句法解析的神经网络结构, 首先通过词向量表达, 将每个词转换为 d 维向量, $e_i^w \in \mathbf{R}^d$ 。全部词嵌入的向量矩阵为 $\mathbf{E}^w \in \mathbf{R}^{d \times N_w}$ 。其中, N_w 是词典的大小。然后, 模型还映射了一个词性标签和一个弧标签到一个 d 维的向量空间, 其中, $e_i^t, e_j^l \in \mathbf{R}^d$ 表示第 i 个词性标签和第 j 个弧标签。相应的, 词性的嵌入矩阵为 $\mathbf{E}^t \in \mathbf{R}^{d \times N_t}$; 弧标签的嵌入矩阵为 $\mathbf{E}^l \in \mathbf{R}^{d \times N_l}$, 这里 N_t 是词性标签的数量, N_l 是弧标签的数量。

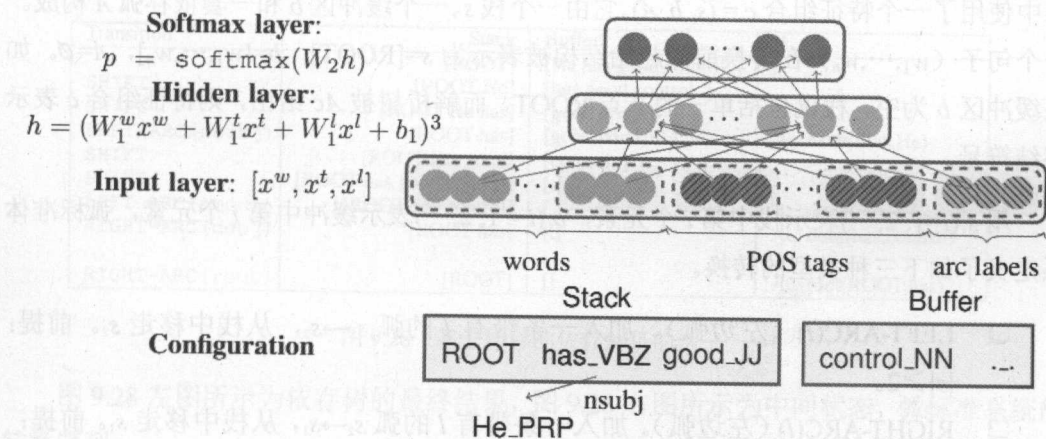


图 9.27 依存解析的神经网络架构

网络模型是一个具有一个输入层、一个隐含层和一个输出层的 LSTM 神经网络, 输入层由 S^w 、 S^t 和 S^l 三部分共同构成, 符号 n_w 、 n_t 、 n_l 表示每种输入类型的长度。这样, 加入 $x^w = [e_{w_1}^w, e_{w_2}^w, \dots, e_{w_{n_w}}^w]$ 到输入层, 其中, $S^w = \{w_1, \dots, w_{n_w}\}$, 与之相似, 词性标签特征 x^t 和弧标注特征 x^l 也被加入到层中。

在输入层到隐含层的映射阶段, 隐含层是一个 d 维节点, 激活函数是 cube 函数。

$$h = (W_1^w \cdot x^w + W_1^t \cdot x^t + W_1^l \cdot x^l + b_1)^3$$

其中, $W_1^w \in \mathbf{R}^{d_h \times (d \cdot n_w)}$ 、 $W_1^t \in \mathbf{R}^{d_h \times (d \cdot n_t)}$ 、 $W_1^l \in \mathbf{R}^{d_h \times (d \cdot n_l)}$ 是权重, $b_1 \in \mathbf{R}^{d_h}$ 是偏置项。

在隐含层顶部使用 softmax 函数进行分类。该函数用于生成多类的概率 $p = \text{softmax}(W_2 h)$, 这里 $W_2 \in \mathbf{R}^{T \times d_h}$ 。

2. 特征组合

1) 基于转移的识别过程

深度学习依存解析器在神经网络中嵌入了词性标记和弧标签。虽然词性标注 $p = \{\text{NN},$

NNP, NNS, DT, JJ, ...}和弧标签 $L=\{\text{amod, tmod, nsubj, csubj, dobj, ...}\}$ 都是离散的集合,但它们如同词汇一样都展示出很大的语义相似性。例如, NN (单数名词) 与 NNS (复数名词) 比 DT (限定词) 要更接近, 或者 amod (形容词修饰成分) 与 num (数量词修饰成分) 应该比 nsubj (名词性主语) 更接近。我们希望这些词性和弧的语义特征对获取最终的输出是有效的。

句法解析的识别过程, 称为基于转移 (Transition-Based) 的依存句法识别, 基本上都采用了弧标准体系 (Arc-Standard System), 这是最流行的一种转换系统。在弧标准体系中使用了一个特征组合 $c=(s, b, A)$ 。它由一个栈 s 、一个缓冲区 b 和一套依存弧 A 构成。一个句子 (w_1, \dots, w_n) 在转换前的起始结构被表示为 $s=[\text{ROOT}]$, $b=[w_1, \dots, w_n]$, $A=\emptyset$ 。如果缓冲区 b 为空, 栈仅包括单一的节点 ROOT, 而解析树被 A_c 给出, 则特征组合 c 表示终结符号。

用 $s_i(i=1, 2, \dots)$ 表示栈中第 i 个元素, $b_i(i=1, 2, \dots)$ 表示缓冲中第 i 个元素, 弧标准体系定义了如下三种类型的转换。

- ❑ LEFT-ARC(l) (左边弧)。加入一条带有 l 的弧 $s_1 \rightarrow s_2$, 从栈中移走 s_2 。前提: $|s| \geq 2$ 。
- ❑ RIGHT-ARC(l) (右边弧)。加入一条带有 l 的弧 $s_2 \rightarrow s_1$, 从栈中移走 s_1 。前提: $|s| \geq 2$ 。
- ❑ SHIFT (转换)。从缓冲移动 b_1 到栈。前提: $|b| \geq 1$ 。

在解析器标注过程中, 转换次数共有 $|T|=2N+1$ 次。其中, N 是大量的互不相同的弧标签。表 9.5 展示了基于转移算法的特征模板。

表 9.5 依存树的转换特征

单个词特征 (9)	词对 (两个) 特征 (8)	三词特征 (8)
$s_1 \cdot w;$	$s_1 \cdot wt \circ s_2 \cdot wt;$	$s_2 \cdot t \circ s_1 \cdot t \circ b_1 \cdot t;$
$s_1 \cdot t;$	$s_1 \cdot wt \circ s_2 \cdot w;$	$s_2 \cdot t \circ s_1 \cdot w \circ b_1 \cdot t;$
$s_1 \cdot wt;$	$s_1 \cdot wt \circ s_2 \cdot t;$	$s_2 \cdot t \circ s_1 \cdot t \circ lc_1(s_1) \cdot t;$
$s_2 \cdot w;$	$s_1 \cdot w \circ s_2 \cdot wt;$	$s_2 \cdot t \circ s_1 \cdot t \circ rc_1(s_1) \cdot t;$
$s_2 \cdot t;$	$s_1 \cdot t \circ s_2 \cdot wt;$	$s_2 \cdot t \circ s_1 \cdot t \circ lc_1(s_2) \cdot t;$
$s_2 \cdot wt;$	$s_1 \cdot w \circ s_2 \cdot w;$	$s_2 \cdot t \circ s_1 \cdot t \circ rc_1(s_2) \cdot t;$
$b_1 \cdot w;$	$s_1 \cdot t \circ s_2 \cdot t;$	$s_2 \cdot t \circ s_1 \cdot w \circ rc_1(s_2) \cdot t;$
$b_1 \cdot t;$	$s_1 \cdot t \circ b_1 \cdot t;$	$s_2 \cdot t \circ s_1 \cdot w \circ lc_1(s_1) \cdot t;$
$b_1 \cdot wt;$		

表 9.5 展示了一个用于分析弧 $lc_1(s_i)$ 和 $rc_1(s_i)$ 的特征模板, $lc_1(s_i)$ 和 $rc_1(s_i)$ 表示 s_i 的最左和最右的孩子节点, w 表示词, t 表示词性标签。

下面给出一个例子, 显示出从初始的特征组合到转换终止的过程, 如图 9.28 所示。

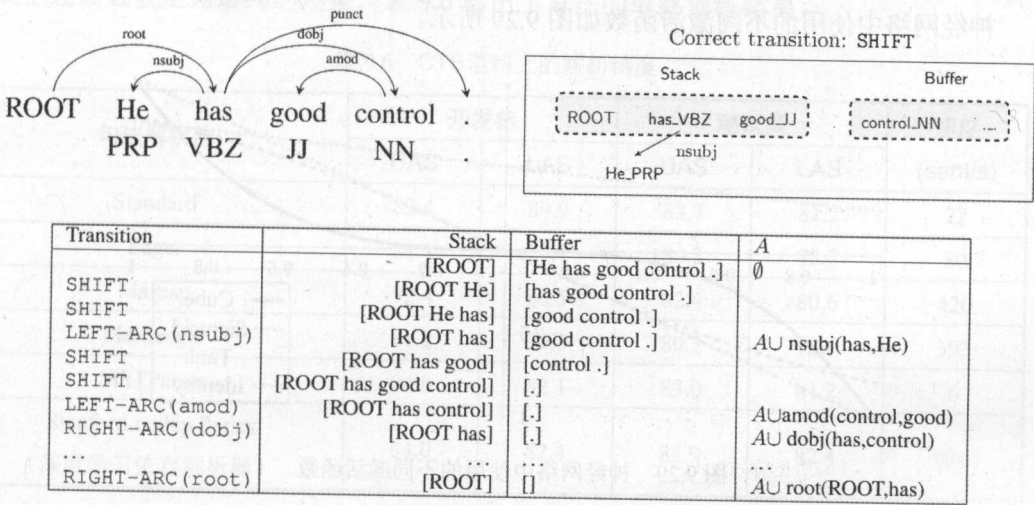


图 9.28 基于转移的依存解析的例子

图 9.28 左图所示为依存树的最终结果, 图 9.28 右图所示为中间状态, 弧标准系统的转换序列。

2) S^w 、 S^t 、 S^a 的选择

深度学习依存解析的特征模板是一组基于“栈/缓冲区 (Stack / Buffer)”位置的元素, 包括词汇、词性和弧标签的信息。这些信息对预测是非常有用的。解析器用 S^w 、 S^t 、 S^a 三个符号来分别表示词汇、词性和弧标签的信息。在网络模型中, 依存解析器使用了庞大的标签集: S^w 的 n_w 中包含了 18 个元素。

(1) 在栈和缓冲区上的前三个词: s_1 、 s_2 、 s_3 、 b_1 、 b_2 、 b_3 。

(2) 在栈顶前两个词的第一个和第二个最右侧/最左侧的孩子节点: $lc_1(s_i)$ 、 $rc_1(s_i)$ 、 $lc_2(s_i)$ 、 $rc_2(s_i)$, $i=1,2$ 。在栈顶前两个词的最右侧/最左侧的孩子节点的最右侧/最左侧的孩子节点: $lc_1(lc_1(s_i))$ 、 $rc_1(rc_1(s_i))$, $i=1,2$ 。

(3) 与之相一致, 词性标签也是 18 个, 弧标签使用了 12 个 (需要排除栈/缓冲区的 6 个词)。

该解析器的一个优势在于, 可以很容易地添加一组丰富标签元素, 而无须手工编写更多的特征函数 (指示函数)。

例如，上述给定 $S'=\{lc_1(s_2).t, s_2.t, rc_1(s_2).t, s_1.t\}$ 的特征组合中，从语料中按顺序抽取出 PRP、VBZ、NULL、JJ。这里使用一个特殊的标签 NULL 代表不存在的元素。

3) Cube 激活函数

神经网络中使用的不同激活函数如图 9.29 所示。

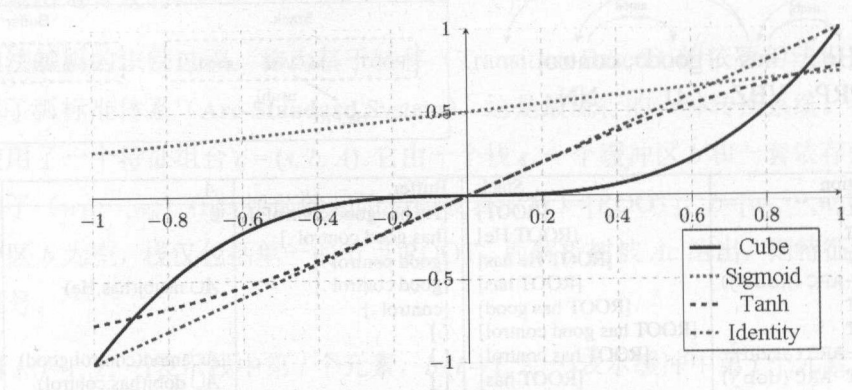


图 9.29 神经网络中使用的不同激活函数

深度学习的依存解析模型中引入一个新的激活函数：Cube $g(x)=x^3$ ，以代替常用的 Tanh 和 Sigmoid 函数。直觉上，使用 $g(x)=x^3$ ，能为像 x_i 、 x_j 、 x_k 这样的三个不同元素的连乘积形式建立更好的模型。

$$g(w_1x_1 + \dots + w_mx_m + b) = \sum_{i,j,k} (w_iw_jw_k)x_ix_jx_k + \sum_{i,j} b(w_iw_j)x_ix_j \dots$$

在此模型中， x_i 、 x_j 、 x_k 是具有不同维度的三个嵌入结构，三个元素的相互作用是依存分析的非常想要得到的性质。实验结果也验证了 Cube 激活函数的成功。然而，这种激活函数的表现力仍然处于理论上的调查阶段。

3. 损失函数与训练

通过依存树库的语料集中生成训练样本 $\{(c_i, t_i)\}_{i=1}^m$ 。其中， c_i 是一个特征组合， $t_i \in T$ 是转换次数。最终的训练目标是最小化交叉熵损失加上一个 L2-正则项。

$$L(\theta) = -\sum_i \log p_{t_i} + \frac{\lambda}{2} \|\theta\|^2$$

这里 θ 代表了所有参数 $\{W_1^w, W_1^t, W_1^l, b_1, W_2, E^w, E^t, E^l\}$ 。一个细微变化是，算法只在实际可行的转换间计算了 softmax 的概率。

算法使用预训练的词向量来初始化参数 E^w ，词向量的维度为 50 维，该词向量来自中文维基百科的语料，使用 $(-0.01, 0.01)$ 随机数来初始化 E^t 和 E^l 。

在训练阶段，训练误差的导数将被反向传播到嵌入向量中，算法使用了最小批次 AdaGrad (mini-batched AdaGrad) 进行优化，丢包率设置为 0.5。

在训练中，还使用了一个开发集进行验证，设置上述参数的网络取得了最优的得分，因此上述参数被定为最终的结果。表 9.6 给出了算法的最终训练结果。

表 9.6 CTB语料上的解析精度

句法解析器	开发集		测试集		速度 (sent/s)
	UAS	LAS	UAS	LAS	
Standard	82.4	80.9	82.7	81.2	72
Eager	81.1	79.7	80.3	78.7	80
Malt:sp	82.4	80.5	82.4	80.6	420
Malt:eager	81.2	79.3	80.2	78.4	393
MSTParser	84.0	82.1	83.0	81.2	6
Stanford nndep Parser (深度学习依存解析器)	84.0	82.4	83.9	82.4	936

9.4.4 Stanford 依存解析的训练过程

在充分了解依存句法解析器的原理之后，本节继续通过一个实例来介绍 Stanford 依存句法解析器的训练过程。通过这个过程，读者可以使用自己的数据训练一个新的解析器，数据使用 CoNLL-X 版本的数据格式（有关此种数据格式的细 节，第 6 章已经介绍过）。

不幸的是，这个过程并不简单，整个训练过程占用大量的 CPU 资源，耗时很长，而且需要读者已经具有支持 Stanford 标注规范的汉语树库资源。我们使用的是宾州短语结构树库（CTB 8.0）。整个树库已经被加载到数据库中。读者可以从自己的数据库中导出该树库。导出的文件格式如图 9.30 所示。

```
( (IP-HLN (NP-SBJ (NP-PN (NR 上海) (NR 浦东)) (NP (NN 开发)
( (FRAG (NN 新华社) (NR 上海) (NT 二月) (NT 十日) (NN 电
( (IP (NP-PN-SBJ (NR 上海) (NR 浦东)) (VP (VP (LCP-TMP (NI
( (IP (IP (NP-SBJ (NP-PN (NR 浦东)) (NP (NN 开发) (NN :
( (IP (PP (P 对) (NP (PN 此))) (PU .) (NP-PN-SBJ
( (IP (NP-SBJ (CP (WHNP-4 (-NONE- *OP*)) (CP (IP (NP-SBJ (-NONE-
( (IP (IP (NP-SBJ (NN 建筑)) (VP (VC 是) (NP-PRD (CP-APP
( (IP (IP (PP-PRP (P 为) (IP (NP-SBJ (-NONE- *PRO*)) (
( (IP (IP (NP-SBJ (NN 建筑) (NN 公司)) (VP (VV 进)
( (IP (IP (CP-ADV (ADV (CS 尽管)) (IP (NP-TPC (CP (WHNP-1 (-NOI
( (FRAG (PU ( ) (VV 完) (PU ) ) ) ) )
```

图 9.30 导出的文件格式

为了便于后续的程序处理，树库中短语间的换行符都被删掉了，整个句法树变为单独的一行。这样，一行就代表了一个完整的句子。在下面的训练过程中，假设读者已经拥有了上述结构的宾州短语结构树库。这样，整个训练过程大致分为如下三个阶段。

1. 将树库转换为 CoNLL-X 版本的形式，并划分出训练集和开发集两部分

1) 将短语结构树库转换为 CoNLL-X 版本形式

回顾一下在第 6 章训练短语结构树库的过程。我们曾经使用过名为 `StanfordParser` 的 Python 类。现在，在这个类中加入一个新的方法，代码如下。

```
...
def penn2conlldep(self, treebank, conllfile):
    self.trainline = 'java -mxlg -cp "' + self.jarpath + '" edu.stanford.nlp.
trees.international.pennchinese.ChineseGrammaticalStructure -basic -keepPunct -conllx
-treeFile "' + treebank + '" > "' + conllfile + '"'
    os.system(self.trainline)
    print "save model to ", conllfile
```

该函数的功能是将短语结构树库转换为依存树库结构。执行代码如下。

```
# -*- coding: utf-8 -*-
import sys, os
from stanford import *
from framework import *
reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
os.environ['JAVA_HOME'] = 'D:/Java/jdk1.8.0_91/bin/java.exe' # 配置环境变量
treebank = "treebank/ctb8_all_in_one.txt" # 宾州树库文件的文本形式——导出自数据库
root = "E:/nltk_data/stanford-corenlp/" # 安装库
parser=StanfordParser(root)
treeconll = "treebank/ctb8_all_dep.conll"
parser.penn2conlldep(treebank, treeconll)
```

执行结果如图 9.31 所示。

在处理过程中可能会出现几处错误，正常情况下，应不超过 10 处，程序提示可以自己修正，这里忽略不计。现在，我们有能力把整个树库转换为一个完整的 CoNLL 格式的文本，但我们的工作还未最终完成。

```

1 上海> _NR NR _2 nn _ _
2 浦东> _NR NR _6 nn _ _
3 开发> _NN NN _6 conj> _ _
4 与> _CC CC _6 cc> _ _
5 法制> _NN NN _6 nn _ _
6 建设> _NN NN _7 nsubj> _ _
7 同步> _VV VV _0 root> _ _

1 新华社> _NN NN _7 dep> _ _
2 上海> _NR NR _7 dep> _ _
3 二月> _NT NT _7 dep> _ _
4 十日> _NT NT _7 dep> _ _
5 电> _NN NN _7 dep> _ _
6 (> _PU PU _7 punct> _ _
7 记者> _NN NN _0 root> _ _
8 谢金虎> _NR NR _7 dep> _ _
9 、> _PU PU _7 punct> _ _
10 张持坚> _NR NR _7 dep> _ _
11 )> _PU PU _7 punct> _ _

```

图 9.31 执行结果

2) 现在要划分出训练集和开发集两部分

首先编写一个按比例随机抽取样本的函数，代码如下。

```

# 分割训练集和开发集
def splitcorpus(rawlist, ratio):
    rawlen = len(rawlist)
    devlist = random.sample(rawlist, int(rawlen*ratio))
    trainlist = [tree for tree in rawlist if tree not in devlist]
    return trainlist, devlist

```

再次执行短语结构到依存树库的转换，代码如下。

```

...
os.environ['JAVA_HOME'] = 'D:/Java/jdk1.8.0_91/bin/java.exe' # 配置环境变量

treebank = "treebank/ctb8_all_in_one.txt"
treelist = readfile(treebank).splitlines()
trainlist, devlist = splitcorpus(treelist, 0.1)
trainpath = "treebank/train.txt"; devpath = "treebank/dev.txt"
savefile(trainpath, "\n".join(trainlist))
savefile(devpath, "\n".join(devlist))
root = "E:/nltk_data/stanford-corenlp/" # 安装库
parser = StanfordParser(root)
train_conll = "treebank/train.conll"

```

```
parser.penn2conlldep(trainpath,train_conll)

dev_conll = "treebank/dev.conll"
parser.penn2conlldep(devpath,dev_conll)
```

这样,整个树库按照 90%和 10%的比例被划分为如下两大部分:训练集——train.conll 和开发集——dev.conll。

2. 将树库中的词汇训练成 Word2Vec 的词向量

有了前面几节 Word2Vec 的基础,这一步就变得比较简单了。将完整版(未分割)依存树库文件直接训练成词向量。代码如下。

```
# -*- coding: utf-8 -*-
import sys,os
from stanford import *
from framework import *
from gensim.models import word2vec
# 设置 UTF-8 输出环境
reload(sys)
sys.setdefaultencoding('utf-8')

trainpath = "treebank/ctb8_all_dep.conll"
sentlist = [];wordlist = []
for tokens in readfile(trainpath).splitlines():
    tokenlist = tokens.split("\t")
    if len(tokenlist) > 1:
        wordlist.append(tokenlist[1])
    else:
        sentlist.append(wordlist)
        wordlist = []

epochs = 20
w2v = word2vec.Word2Vec(workers = 8,sample = 1e-5, size = 50, min_count=1, window = 5)
w2v.build_vocab(sentlist)
for epoch in range(epochs):
    print('epoch' + str(epoch))
    w2v.train(sentlist)
    w2v.alpha *= 0.9
    w2v.min_alpha = w2v.alpha
# w2v.save('treebank/ctb8vector.bin') # 存储为二进制格式
w2v.save_word2vec_format('treebank/vectors.bin.txt', fvocab=None, binary=False)
print("word2vec DONE.")
```


在训练过程中应注意如下内容。

- ❑ Stanford 神经网络句法训练器仅接收文本格式的词向量文件，因此输出的词向量文件必须为文本格式，而不是二进制格式。
- ❑ 因为语料规模不大，建议词向量维度为 50 维，这也与依存解析器所需的词向量默认维度一致。
- ❑ 训练好的文件第一行是词汇总数、向量维度，而在训练依存解析器时，该行不能识别，必须删除。

训练结果如图 9.32 所示。

```
70325 50
. -0.274894 -0.028372 0.147142 0.039184 -0.950229 0.699274 -(
的 -0.311799 0.095197 0.195840 -0.106440 -0.906255 0.850500 -(
。 -0.370230 0.078092 0.143790 -0.020081 -0.791972 1.030957 -(
是 -0.207742 0.026937 0.480517 0.055075 -1.212808 0.311762 -0.
在 -0.296426 -0.164172 -0.241026 -0.608993 -0.393389 0.584489
了 -0.571188 -0.256669 0.597079 -0.460994 -0.196890 0.591653 (
一 0.251492 -0.410072 1.100197 -0.291942 -0.940070 0.891943 -(
、 0.976424 0.896736 -0.449025 0.529967 0.700605 1.310106 0.3!
不 -0.281701 -0.029631 0.481810 0.471837 -1.530925 0.396686 -(
个 0.168122 0.000482 0.937549 -0.439712 -1.031051 0.527684 -0.
```

图 9.32 训练结果

该文件使用时需要删除第一行：70325 50，然后保存。

3. 使用神经网络模型训练依存句法树

继续在 StanfordParser 类中加入训练神经网络模型的新方法，代码如下。

```
def
train_nndeptrain(self, trainpath, devpath, embedpath, embedgsize, modelpath): # 创建模型
文件
    self.trainline = 'java -mx8g -cp "'+self.jarpath+'" edu.stanford.nlp.
parser.nnep.DependencyParser -tlp edu.stanford.nlp.trees.international.pennchinese.
ChineseTreebankLanguagePack '
    self.trainline += '-trainFile "'+trainpath+'" '
    self.trainline += '-devFile "'+devpath+'" '
    self.trainline += '-trainingThreads 4 '
    self.trainline += '-embedFile "'+embedpath+'" '
    self.trainline += '-embeddingSize '+ embedgsize+' ' # 去掉文件 txt 第一行
的词汇和维度信息：70325 50
    self.trainline += '-model "'+modelpath+'"'
```

```
print self.trainline
os.system(self.trainline)
print "save model to ",modelpath
```

限于篇幅，这里不一一给出每个参数的解释，建议在设置时，读者一定要参考 Stanford 网站上有关神经网络解析器的说明（网址：<http://nlp.stanford.edu/software/nndep.shtml>），了解每个参数的意义，并根据自己的设备、数据来设置符合自己的参数。

最后，执行代码如下。

```
# -*- coding: utf-8 -*-
import sys,os
from stanford import *

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
os.environ['JAVA_HOME'] = 'D:\\Java7\\jdk1.8.0_65\\bin\\java.exe' # 配置环境变量

root = "E:/nltk_data/stanford-corenlp/" # 安装库
trainpath = "treebank/train.conll"
devpath = "treebank/dev.conll"
embedpath = "treebank/vectors.bin.txt"
embedsize = "50"
modelpath = "treebank/nndep.model.txt.gz"
parser=StanfordParser(root)
parser.train_nndep(trainpath,devpath,embedpath,embedsize,modelpath)
```

执行结果：

训练非常耗时，大约要跑三天时间，具体时间视设备配置情况而定，输出模型：
nndep.model.txt.gz。

9.5 结语

本章从神经网络的理论开始，系统地介绍了深度学习在 NLP 上的应用。这些应用包括在词汇语义上的突破——Word2Vec 算法、使用 LSTM 进行序列标注、使用 LSTM 解决依存句法问题。

首先详细介绍了神经网络的算法架构，并给出了最基本的神经网络：Sigmoid 梯度

下降法的算法推导和源码。然后简要介绍了 BP 神经网络的算法架构和反向传播的概念，以及反向传播的推导过程。

接下来，详细讲解了分布式词向量表达的概念，使用词向量表达的实现，以及 Word2Vec 的算法框架。在 Word2Vec 中，给出了在 CBOW 模型，Skip-Gram 模型的网络结构，以及使用 Hierarchical Softmax 和负采样的学习过程。最后还给出了一个使用 Word2Vec 生成词向量的案例。

在 NLP 与 RNN 一节，从简单的 RNN 开始，首先给出了 RNN 的网络架构和推导过程，特别是在反向传播阶段，给出了详细的说明，之后，通过代码片段，给出了该网络的一个简单实现。LSTM 网络是对简单的 RNN 的一种改进。通过增加三个控制门：输入门、遗忘门和输出门来控制状态的更新，有效地避免梯度消失或梯度爆炸的问题。对于这三个门的工作过程，我们使用一个简单的 LSTM 代码向读者展示出来。读者可以通过简单的调试即可了解 LSTM 的工作过程。

在最后一节，讲解了如下两个真实的案例。

(1) 使用 Word2Vec 和 LSTM 实现序列标注——中文分词的全过程。

(2) 使用 Stanford-NLP 的基于转移的 LSTM 网络做的依存句法分析。

第一个案例实现了 94% 的精度。第二个案例，Stanford-NLP 实现了 84% 的精度。强有力地证明了深度学习对 NLP 的重要意义。

第 10 章

语义计算的架构

自然语言处理理论从诞生、发展直至走向成熟，是一个漫长而又艰苦的过程。最早，NLP 的基本任务在于正确地解析句子的语法结构，认为只有完整地解析句子的语法结构，才能在后续的机器翻译环节中正确转换和生成目标语言的句子，以完成机器翻译上的全过程。人们对句法的完全解析经历了漫长的过程，却始终没有达到预期的效果。人们在反思中发现，不同语言的语法逻辑虽然有相似性，但差异性也是明显的，硬性的匹配往往很难达到预期的效果。一个完整的 NLP 应用，不可能绕开语义识别这个环节。

认知语言学产生的目的之一就是要解释什么是语义，语言中构成语义的要素有哪些，以及如何能够正确地识别出语义。迄今为止，NLP 在句义的识别上虽然仍有待于进一步发展，但已经获得了巨大的成功。

本章是本书的最后一章。这一章提出了一些结论性的观点，即为完整句义的解析提供了一个现实可行的架构，并为读者呈现句子语义识别的基本框架和已经实现的实例。

10.1 句子的语义和语法预处理

前面各个章节讲解了很多 NLP 的功能模块及实现的算法。每个模块都处理不同的问题，包括词法分析模块：中文分词、词性标注、命名实体识别、语义组块等；句法分析模块：短语结构、依存句法分析；认知计算模块：隐喻计算、Word2Vec 词向量、语义角

色标注。上述模块构成了 NLP 的句子语义解析的核心系统。

从本节开始介绍一些新的模块,包括:复句切分和指代消解。这些模块都不同程度地用于语义的预处理工作。

复句切分与关系识别,是将复句切分为单句的形式并标注出切分后的单句之间的相互依存关系。指代消解是确定文本中的名词和代词短语所指向的真实实体的过程,主要包含人称代词消解和名词短语消解。这两个模块都属于认知领域中对事件集合的压缩和省略的一种恢复,属于认知语言学中的完形范畴。

本节介绍的有些模块刚刚发展起来,其算法和语料资源都尚不成熟;有些则已经达到了实用的精度。本节的任务是将上述所有模块整合起来,形成一个完整的 NLP 语义解析系统。

10.1.1 长句切分和融合

一般来讲,汉语句子的长度越长,其结构就越复杂。汉语是一种表义的语言,汉语句子的组织和印欧语言有很大的不同。语言中常用的断句符(句号、感叹号等),不仅表示一个句子的结束,在书面语中更多地来表示一系列事件的结束。这些事件常有相同或相似的语境(Context),构成一个句群。重复使用的概念在句群中可以被压缩或省略,这在后面的指代消解中体现得很明显,这里就不重复了。而汉语中逗号这类非断句符,除表示小句和短语停顿,也用来分割句群中的单句,有助于语义的辨析。

这种情况对句法解析的影响非常大,汉语句子的长度越长,句法分析的准确率就越低。例如,CTB 5.0 的句子平均长度是 27 个词,目前依存句法分析的最好水平为 80%;中国台湾中央研究院的 Sinica Treebank 句子平均长度为 6 个词,而同样算法分析的准确率达到 90%。因此,正确地区别出那些非断句符的断句作用,对句法分析的精度具有巨大的提升作用。

为了解决长句切分的问题,很多科研机构都做了研究,其中比较著名的是哈工大社会计算与信息检索研究中心所做一系列实验,实验结果比较成功。实验分为如下两个阶段。

1. 切分阶段

我们使用的标注规则,来自《基于统计方法的汉语依存句法分析研究》(马金山)一文中标注规则。文中将以逗号、冒号、分号、句号、问号和叹号结尾的字符串称为一个

片段。在本文所使用的 863 词性标记集中, 这些标点被统一标识为 wp。根据片段的语法结构, 本文制定了一个分类标准, 将片段分为不同的类别, 类别值通过片段末尾的标点进行标识。片段的类别共分为如下 5 种。

(1) 分句。分句是语法结构完整的片段, 分句之间只有语义上的联系, 在句法结构上没有联系, 标识的方法是将片段末尾标点的词性标注为 wp1, 如:

香港/ns 中华/nz 总商会/n 今天/nt 上午/nt 举行/v 会员/n 元旦/nt 团拜/n 酒会/n , /wp1 新华社/ni 香港/ns 分社/n 副/b 社长/n 秦文俊/nh 出席/v 了/u 酒会/n 。/wp1

(2) 无宾语结构——宾语是一个从句。片段的谓语是及物动词, 但是谓语和宾语之间被标点间隔, 将该结构末尾的标点标识为 wp3, 如:

我们/r 在/p 调研/n 过程/n 中/nd 了解/v 到/v , /wp3 目前/nt 我国/n 企业/n 集团/n 的/u 发展/n 总体/n 上/nd 是/v 比较/d 顺利/a 的/u , /wp1 但/c 存在/v 一些/m 值得/v 注意/v 的/u 问题/n 。/wp2

(3) 无主语结构——主语脱落的情况。片段的语法结构不完整, 主语被省略或者位于前面的片段之中。将该结构末尾的标点标识为 wp2, 如:

按/p 保护价/n 敞开/d 收购/v , /wp2 充分/d 掌握/v 粮源/n , /wp2 才/d 能/v 保护/v 农民/n 的/u 利益/n , /wp2 才/d 能/v 做到/v 顺价/d 销售/v , /wp2 才/d 不/d 会/v 被/p 粮贩子/n 牵/v 着/u 鼻子/n 走/v 。/wp2

(4) 短语。片段由一个句法成分构成, 是一个短语或者一个词, 片段中无谓语动词。通常是名词短语、介词短语或者连词。将该结构末尾的标点标识为 wp4, 如:

今日/nt 清晨/nt , /wp4 在/p 嘹亮/a 的/u 国歌声/n 中/nd , /wp4 拉萨/ns 隆重/d 举行/v 升/v 国旗/n 仪式/n 。/wp1

(5) 停顿语。片段由两个或以上的句法成分构成, 并且在句法结构和语义上均不完整, 标点只起停顿的作用。将该结构末尾的标点标识为 wp5, 如:

双方/n 应/v 在/p 此/r 基础/n 上/nd , /wp5 妥善/a 处理/v 两/m 国/n 间/nd 存在/v 的/u 问题/n 。/wp3

——《基于统计方法的汉语依存句法分析研究》

现代汉语的书面语中, 虽然长句通过这样的标注规则被分割为多个片段, 但每个片段的长度都不大。而 Sinica Treebank 中的句子就是经过类似标准分割之后的片段, 所以长度很小。应该说上述切分标准还是比较合理的。在进行句法分析之前, 第 (1)、(2)

种情况必须进行切分，切分出的结果为两个分句，或一主一从，它们之间的关系是句子与句子之间的关系。第（4）、（5）种情况，是短语关系而不是句子间的关系，所以在句法分析之前，不需要切分。对于第（3）种情况，如果指代消解能够识别出脱落的主语，那么需要切分；如果没有指代消解的过程，那么需要为这种特殊的句式单独训练样本，以提高依存解析的精度。

这样，该问题就转变为一个多分类的机器学习问题。大量的实验证明，长句切分为单句最有效的算法是支持向量机（SVM）分类器。支持向量机是一种基于统计学习理论的常用机器学习算法之一，具有较好的推广能力和非线性处理能力，尤其在处理高维数据时，有效地解决了“维数灾难”问题，在人脸检测、网页分类、手写体数字识别、图像检索等领域应用广泛。

有关支持向量算法的更多细节读者可以参考有关机器学习类的相关读物，这里推荐给大家一个著名的 SVM 框架——Libsvm，读者可从 <https://www.csie.ntu.edu.tw/~cjlin/libsvm/> 下载。该框架是台湾大学林智仁（Lin Chih-Jen）教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包，它不但提供了编译好的可在 Windows 系统执行的文件，还提供了源代码，方便改进、修改，以及其他操作系统上应用。该软件对 SVM 所涉及的参数调节相对比较少，提供了很多的默认参数，利用这些默认参数可以解决很多问题；并提供了交互检验（Cross Validation）的功能。

针对多分类的要求，哈工大社会计算与信息检索研究中心还给出了每个类别的特征，这些特征取自于经过分词和词性标注的片段。分类器所使用的特征如表 10.1 所示：

表 10.1 分类器所使用的特征

特 征	描 述	特 征	描 述
len	片段长度是否大于4	shi	是否含有“是”
vg	是否含有一般动词	you	是否含有“有”
vx	是否含有系动词	zhe	是否含有“着、了、过”
vt	是否含有及物动词	vg_n	是否含有动宾搭配
vi	是否含有不及物动词	p_nd	是否有介词和方位词的组合
nd	末尾是否为方位名词	tag1	第一个词的词性
de	是否含有“的”	tag_l	最后一个词的词性
dei	是否含有“得”	punct	片段末尾的标点

训练数据使用的是 1998 年上半年的人民日报树库，该树库共有 212 527 个词、1 万个句子，每个句子的平均长度为 21.3 个词。实验以前 8 000 个句子作为训练数据，8 000～

9 000 个句子作为开发集，最后 1 000 个句子作为测试集。表 10.2 所示为 5 种片段在语料中的分布情况。

表 10.2 5 种片段在语料中的分布情况

片段类型	wp1	wp2	wp3	wp4	wp5
片段数量	7 838	8 975	2 367	3 673	943

分类器使用的是 LibSVM，测试结果如表 10.3 所示。

表 10.3 片段类型的识别结果

	wp1	wp2	wp3	wp4	wp5
准确率	0.762 3	0.805 6	0.788 7	0.885 7	0.391 8
召回率	0.789 3	0.878 8	0.584 0	0.781 5	0.400 0
F-Score	0.775 6	0.840 6	0.671 1	0.830 4	0.395 8

表 10.3 中，wp5 的片段类型识别准确率较低，主要是因为该片段在句子中的比例较少，特征数据不足，导致分类器在识别的时候产生困难。由于不对 wp5 进行切分，统计意义不大。

之后，根据片段识别的结果，将句子进行分割。分割的位置选择为 wp1、wp2、wp3、wp4。但依照本文的观点，是否对 wp4 也进行切分，还有待商榷，暂且依照文献中的切分规则进行下面步骤。

2. 合并阶段

由于切分之后的片段在语法上是独立的，都可作为句法分析的基本单元，即只有一个核心词与外部发生联系。也就是说，各个片段只有一个核心节点，那么只要分析出各个片段的节点之间的依存关系，即可将句子合并，这样就能够得到完整的依存句法树。

需要说明的是，片段之间的关系与片段内部的关系不同，片段内部的依存关系主要是词与词之间的关系，片段间的依存关系则主要是短语（wp4）或句子（wp1、wp2、wp3）之间的关系。片段之间的依存关系类型如表 10.4 所示。

表 10.4 片段之间的依存关系类型

关系类型	描 述	关系类型	描 述
IC	独立分句	VV	连动结构
SBV	主谓结构	ADV	状中结构
VOB	动宾结构	CNJ	关联结构

在标注阶段，每个片段都被分配一种专门的标签。这样该问题又转化为一个多元分类问题，仍然使用 SVM 分类器进行识别。与切分阶段不同的是，此时每个片段都经过了句法分析，已经获得内部的依存关系，可以使用句法信息（节点和弧）作为特征。从片段中抽取的特征如表 10.5 所示。

表 10.5 片段间依存关系识别的上下文特征

特 征	描 述	特 征	描 述
len	片段长度是否大于4	root	根节点的词性
lchild	根节点与最左侧孩子的关系	subj	片段中是否含有主语
rchild	根节点与最右侧孩子的关系	obj	片段中是否含有宾语
shi	根节点是否为“是”	punct	片段末尾标点
you	根节点是否为“有”	position	根节点是否在片段首部或尾部

表 10.5 中，每个特征分别从两个片段中抽取，每个关系类型对应的特征向量共有 20 个。获得片段之间的依存关系之后，根据关系类型将两个片段的核心节点进行连接，形成一棵完整的依存分析树。

确定了上述特征，即可使用 SVM 算法识别片段之间的关系类型，仍然将 Libsvm 作为分类器。各个关系的识别结果如表 10.6 所示。

表 10.6 片段间依存关系的识别结果

	ADV	CNJ	IC	SBV	VOB	VV
准确率	0.877 2	0.692 3	0.728 2	0.727 3	0.691 5	0.647 6
召回率	0.823 0	0.782 6	0.579 2	0.631 6	0.596 3	0.916 8
F-Score	0.849 3	0.734 7	0.645 2	0.676 1	0.640 4	0.759 1

经过片段合并之后，得到完整的依存分析结果，并连接成整句依存分析结果。经过统计，关系准确率、搭配准确率和句子核心词三个指标对结果进行评价，结果如表 10.7 所示。

表 10.7 整句的依存分析结果

	依存关系	依存搭配	核心词
整句准确率	0.645 5	0.695 7	0.701
片段准确率	0.675 7	0.729 2	0.754

表 10.7 中，第 2 行表示未对句子进行片段划分，直接对整句进行依存分析的结果，

第3行是按照本文所描述的基于片段的句法分析所得到的结果。从结果中能够看出,基于片段的分析与整句分析相比,依存句法分析的准确率有了较大的提高,提升了3%~5%。表明基于片段的方法对句法分析的改善具有显著的效果。从分析结果中也能够看出,句子核心词的准确率提高幅度较大,这是因为片段的长度变小,减少了其他词汇的干扰,使句法分析能够更准确地找到句子的根节点。

10.1.2 共指消解

共指消解是确定在文本中哪些名词短语(NPs)指代相同真实世界实体的任务。指代是一类非常复杂的语言现象,它不仅包含人称代词指代和指示代词指代,还包括零指代和一般名词短语之间的指代,每种指代之间存在较大差异,采用单一方法很难解决指代的各种情况,因此只有在对每一种指代现象进行深入研究的基础上,才能够提出较好的解决方法。

指代消解是一个需要知识资源支撑的任务,需要多级语言知识,包括句法知识、语义知识、上下文知识、甚至领域知识,在当前自然语言处理水平下,要有效地得到所需的这些知识仍然不是一件容易的事情,需要一个长期的过程进行逐渐的积累。

汉语指代消解研究的起步较晚,目前的研究还不够深入。相比于英语指代消解的研究,在汉语中指代消解的工作相对较少。汉语和英语两种语言之间的差异,使得直接借鉴英文指代消解的处理方法存在一定的困难,甚至很难确定汉语中哪些指代消解系统是具有代表性的系统。

CoNLL-2012 共享任务中排名第一位的共指消解器是 Stanford NLP 提供的混合“基于规则/机器学习”方法的开源工具,这一方法也称为带有词汇特征的“基于规则的多路筛选(Rule-Based Multi-Pass Sieve)”。其已被证明对机器学习方法很有用。这一指代消解工具具有如下4个特征。

- 指称(Mention)的检测。以前的工作已经表明,抽取高质量的指称(一个共指链中的NP)对指代消解的性能具有重要的作用。指代消解器的性能——召回率和准确率受限于指称检测器。为了提高指称检测器的精度,需要改善指称剪枝的策略,指称剪枝策略决定了消解器的精度。要改善指称检测器的召回,就需要提高句法树中的指称抽取,而指称检测的召回则取决于指称抽取策略和句法解析的质量。

- 预处理。指称检测之后，需要使用如语法分析器和命名实体（NE）识别器等预处理的工具，来计算所提取指称的特征。指代消解器的性能由这些工具产生输出的正确性所决定。
- 指代算法。为了更好地了解前文所说的混合方法，我们重点关注如下三个问题。第一，混合方法是否是必需的，也就是说，如果没有混合方法会怎么样；第二，在多通道筛选方法各筛选器对整体性能做出哪些贡献；第三，筛选器的排序对性能而言有哪些影响。
- 基于分类器的比较。在共享的任务中，Stanford 指代消解器优于普遍使用的指称线对（MP）模式的系统，是一种用于训练确定两个 NP 是否共指的分类器。但是，不能说 Stanford 指代算法优于 MP 模式，因为“多路筛选”并不知道哪个组件对结果做出的贡献最大（如指称检测、特征计算、解算结果）。事实上，许多以前的工作都把重点放在系统比较，而不是模型/方法。

Stanford 指代消解模块使用的训练、开发和测试集是 CoNLL-2012 共享任务数据集，如表 10.8 所示。

表 10.8 训练、开发和测试集的统计结果

	Docs	Mentions（指称）	Chains（链）	Mentions/Chain（指称/链）
训练集	1 391	102 854	28 257	3.6
开发集	172	3 875	14 183	3.7
测试集	166	3 559	12 801	3.6

如表 10.8 中，训练集用于学习概率，开发集为调整阈值，测试集用于评估指代消解器的结果。具体来说，解析器的得分是跨越三个记分程序（MUC、B3 和基于实体的 CEAF） F 值的未加权平均数。值得一提的是，（1）解算器不奖励正确识别的指称。（2）指称被视为正确的，当且仅当有黄金（人工校对）指称和提取指称之间的精确匹配提取。此外，省略代词、系动词和同位语结构被排除在评估之外（官方的共同任务）。

下面给出 Stanford 的两步指代消解器的概述。注意共享任务数据集已经提供了分词和句法解析的结果，需要单独计算。

步骤 1，指称检测，使用两步法构建指称检测器。首先，在抽取阶段，从句法树的所有 NP 和 QP 节点中抽取指称。然后，进行剪枝，识别和过滤掉那些错误的抽取项。算法使用了两种剪枝策略：基于启发式的剪枝和基于学习式的剪枝。启发式是基于规则的，如果是疑问代词、专有名词或数量词，就修剪掉候选指称项。学习式是基于统计的，修剪掉那些概率值低于阈值的指称项。阈值是由开发集确定的。

步骤 2, 基于筛选器的指代消解。筛选器是由一个或多个手工设计规则组成的, 每个筛选器都建立了两个指称的共指关系。指代消解过滤器由一组有序的筛选器构成, 序列根据筛选器的精度排列。首先出现的是最精确的筛子。当给定一个文本时, 指代消解用多个筛选器过滤它: 如果通过第 i 个筛选器, 则使用第 i 个规则来建立共指关系。

指代消解的解算器由 10 个筛选器构成。汉语中心词 (CHM) 筛选器用于识别两个具有相同中心词的指称之间的共指关系。话题处理 (DP) 筛选器用于处理那些第一人称和第二人称的指称情况。抽取字符串匹配 (ESM) 筛选器用于一个非代名词指称到具有相同字符串的指称。精确构造 (PC) 筛选器识别那些基于词汇和语法信息的指称之间的共指现象, 诸如一个指称是否是另一个的缩写, 或者指称是否处于同位结构。此外, 还并入汉语特定规则用于确定一个指称是否是其他命名实体的缩写。严格头匹配 (SHM A-C) 筛选器是三个头词 (中心词) 匹配筛选器, 它包含基于头词匹配的、精度逐渐降低的共指规则。专有头词匹配 (PHM) 筛选器只适用于专有名词的严格头词匹配的宽松版本。代词 (Pro) 筛选器包含从训练集合得到的特征, 用于解算诸如性别和数字等共指现象。最后, 词汇对 (LP) 筛选器用于识别基于词汇特征的指代关系。例如, 一个规则指定两个指称是共指的, 如果其中心词的概率大于 t , 那么其中 t 是使用开发集确定的阈值。

注意: 在任何的筛选器假定两个指称作为共指之前, 通常使用语言约束规则来排除非共指的情况。这些约束被实现为单一的非共指规则, 指定两个指称 m_i 和 m_j , 如果下列五个条件之一成立, 则不能为共指: (1) 它们满足在 “i-within-i” 的约束。(2) 它们指向对话中的不同讲话者, 即使是相同的字符串。(3) 它们在一个系动词结构中。(4) m_i 由两个并列的 NP 构成 (含有 “and”), 而 m_j 是其中之一。(5) m_i 和 m_j 是共指的概率低于某一阈值 (从训练数据计算)。

步骤 3, 后处理。在将其发送到计分程序之前, 需要对指代部分进行后处理。具体来说, 删除 (1) 在同位语结构中, 两个指称之间的所有共指链接, 以及 (2) 孤立的簇。

Stanford CoreNLP 开放了共指消解的程序。算法更多的细节参见 *Deterministic coreference resolution based on entity-centric, precision-ranked rules* (下载地址: http://www.mitpressjournals.org/doi/pdf/10.1162/COLI_a_00152)。Stanford 自然语言处理小组在有关共指消解的网页 (URL: <http://nlp.stanford.edu/software/dcoref.shtml>) 中给出上述算法的评测结果。

Stanford Core NLP v3.6.0 的 dcoref 算法, 在 CoNLL2011 共享任务集中, 获得第一名。系统升级到 v3.6.0 版后使用 v8.01 评分器, 评测数据如图 10.1 所示。

	MUC			B cubed			CEAF (M)			CEAF (E)			BLANC			Avg F1
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	
conllst2011 dev	62.1	59.3	60.7	56.2	48.6	52.1	58.0	57.5	57.8	48.9	53.5	51.1	54.1	47.2	50.1	54.62
conllst2012 dev	65.9	64.1	65.0	58.7	50.9	54.5	59.2	59.6	59.4	48.6	54.3	51.3	59.5	53.7	56.1	56.92

* Automatic mention detection used. Avg F1 = (MUC + B cubed + CEAF)/3.

图 10.1 评测数据

从 Stanford CoreNLP 3.5.2 开始，该系统加入了对汉语共指消解的支持，并在网页中给出了使用说明。由于 3.6.0 版本做了升级，代码与 3.5.2 版本出现较大变化。经过调试后的代码如下。

(1) 将英文的 stanford-corenlp-3.6.0-models.jar 库加入到项目 lib 中，如图 10.2 左图所示为加入 models 之后的 lib 目录。如图 10.2 右图所示为项目当前的 models 子目录，该目录的内容参见第 1 章的配置。

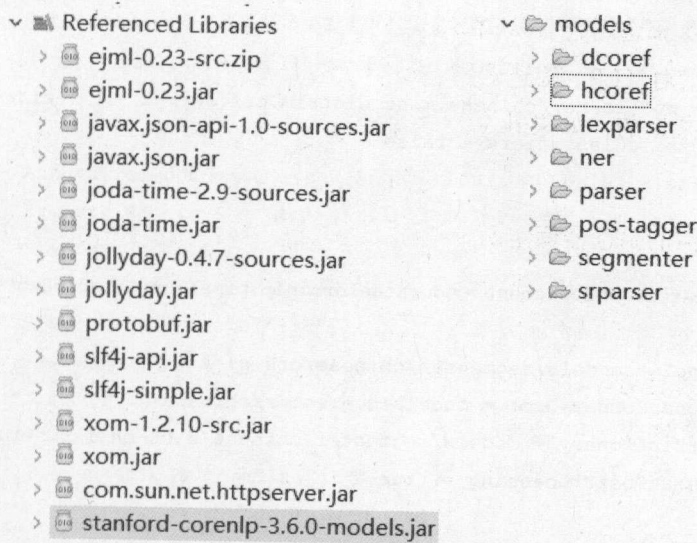


图 10.2 加入 stanford-corenlp-3.6.0-models.jar 后的目录

(2) 修改项目 models/hcoref/properties 目录下的配置文件，文件名为 zh-coref-default.Properties，代码如下。

```
# Pipeline options
annotators = segment, ssplit, pos, lemma, ner, parse, mention, coref

coref.sieves = ChineseHeadMatch, ExactStringMatch, PreciseConstructs, StrictHeadMatch1,
StrictHeadMatch2, StrictHeadMatch3, StrictHeadMatch4, PronounMatch
coref.input.type = raw
```

```

coref.postprocessing = true
coref.calculateFeatureImportance = false
coref.useConstituencyTree = true
#coref.useConstituencyTree = false
coref.useSemantics = false
coref.md.type = RULE
coref.mode = hybrid

coref.path.word2vec =
coref.language = zh
coref.print.md.log = false
coref.defaultPronounAgreement = true
# 这个路径必须指向 stanford-corenlp-3.6.0-models.jar 中的对应文件
coref.big.gender.number = edu/stanford/nlp/models/dcoref/gender.data.gz
# 后面路径都指向本地的 models 模块路径
coref.zh.dict = models/dcoref/zh-attributes.txt.gz

# NER
ner.model = models/ner/chinese.misc.distsim.crf.ser.gz
ner.applyNumericClassifiers = false
ner.useSUTime = false

# segment
customAnnotatorClass.segment = edu.stanford.nlp.pipeline.ChineseSegmenterAnnotator

segment.model = models/segmenter/chinese/ctb.gz
segment.sighanCorporaDict = models/segmenter/chinese
segment.serDictionary = models/segmenter/chinese/dict-chris6.ser.gz
segment.sighanPostProcessing = true

# sentence split
ssplit.boundaryTokenRegex = [.]|[[!]?]+|[. ]|[[!]? ]+

#pos
pos.model = models/pos-tagger/chinese-distsim/chinese-distsim.tagger

#parse
parse.model = models/lexparser/chinesePCFG.ser.gz

```

(3) 共指消解源代码如下。

```

package edu.stanford.nlp.ademo;
import java.util.Map;

```

```

import java.util.Properties;

import edu.stanford.nlp.hcoref.CorefCoreAnnotations;
import edu.stanford.nlp.hcoref.CorefCoreAnnotations.CorefChainAnnotation;
import edu.stanford.nlp.hcoref.data.CorefChain;
import edu.stanford.nlp.hcoref.data.Mention;
import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.util.CoreMap;
import edu.stanford.nlp.util.StringUtils;

public class CorefDemo {

    public static void main(String[] args) {

        String text = "苹果公司很有名，这家公司的人都很努力。";
        args = new String[] {"-props", "models/hcoref/properties/zh-coref-
default.properties" };

        Annotation document = new Annotation(text);
        Properties props = StringUtils.argsToProperties(args);
        StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
        pipeline.annotate(document);
        System.out.println("---");
        System.out.println("coref chains");
        for (CorefChain cc : document.get(CorefCoreAnnotations.CorefChainAnnotation.
class).values()) {
            System.out.println("\t" + cc);
        }
        for (CoreMap sentence : document.get(CoreAnnotations.SentencesAnnotation.
class)) {
            System.out.println("---");
            System.out.println("mentions");
            for (Mention m : sentence.get(CorefCoreAnnotations.CorefMentionsAnnotation.
class)) {
                System.out.println("\t" + m);
            }
        }
    }
}

```


(4) 输出结果如下。

```
---  
coref chains  
---  
mentions  
    苹果 公司  
    这 家 公 司 的 人  
    这 家 公 司
```

10.2 语义角色

10.2.1 谓词论元与语义角色

在句法分析时，我们会遇到如下情况。

杭州大学录取了这名考生。

这名考生杭州大学录取了。

杭州大学把这名考生录取了。

这名考生被杭州大学录取了。

上述这 4 种情况从句法分析上，其生成的句法树结构是不同的，但在语义表达上是完全相同的。可见单纯使用句法解析来实现对语义的获得和运算是不足的。那么，人们需要一种独立于句法树的数据结构来表示语义。

以“动词中心论”为基点的配价理论认为，在一个句子中，动词处于中心地位，动词的从属成分可分为两类：配角成分和说明成分。配角成分参与句子动词的行为，一般以名词性成分出现，配角成分可以分成三种，分别相当于传统语法的主语、直接宾语和间接宾语。说明成分用于说明动词发生的地点、时间、方式和方法。配价中的价反映了动词对其他词项的支配能力，具有不同的支配能力的动词有不同的价。但在确定价的问题上，语言学出现了争论。

后来人们发现，配价理论与谓词论元对句子语义的解释思想相一致。论元 (Argument) 是函数中的一个概念，数学上也称为自变量。函数是数学与逻辑学的基本概念，也是形式语义学的基本概念。函数可简单地表示为 $y=f(x)$ 。其中， y 是因变量， x 是自变量， f

是函子 (Functor)。在形式语义学中, 函子常常称作谓词 (Predicate), 自变量称作论元, 因变量称作开放句 (Open Sentence)。逻辑学上的“一元谓词”、“二元谓词”、“三元谓词”就是从谓词所联系的论元的数目来讲的。方立曾经指出: “谓词用于说明个体的性质或两个及两个以上个体之间的关系。这也就是说, 还应有三元谓词、四元谓词等。事实就是如此, 不过一般的逻辑书都把讨论的范围局限在一元谓词和二元谓词。” 从数理逻辑中引进的术语“论元”相当于“价”的概念。

谓词所支配的论元在句子中所起到的作用各不相同, 表达的语义也不同, 有的作为主体凸显出来, 是谓词支配的必需成分; 有的则是背景、可选的成分。就如同一个场景中, 不同的人所扮演的角色也不同。这些论元就称为语义角色, 如施事、受事、感事等都属于由谓词支配的核心论元。这些论元与谓词一起构成了一个谓词的命题框架。有些是可选的成分, 如处所、时间、方式、工具等。

汉语中常见的语义角色如表 10.9 所示。

表 10.9 汉语中常见的语义角色

属性	语义角色	标注符号	含 义	例句 (黑斜体字为标注的论元)
核心论元	施事、感事	Arg0	<p>参与者是动作的主动发出者, 也可由介词“被”(叫、让、给、由、归)等引进。</p> <p>可以作为及物动词的主语和一类不及物动词。</p> <p><input type="checkbox"/> 决定参与事件或状态。</p> <p><input type="checkbox"/> 造成其他论元的事件或状态的改变。</p> <p><input type="checkbox"/> 相对于其他论元的位置移动</p>	大家都拥出了教室。(大家) 路障被工人搬走了。(工人) 物业的事归她管。(她)
	受事	Arg1	<p>参与者是动作的承受者, 也可由介词“把”(将)引进。</p> <p>这些论元有原施事属性, 这意味着这些论元:</p> <p><input type="checkbox"/> 经历状态变化。</p> <p><input type="checkbox"/> 其因果关系受到其他论元的影响。</p> <p><input type="checkbox"/> 是固定相对于其他论元的运动</p>	张三打破了窗户。 窗户被张三打破了。 他把书还回去了

续表

属性	语义 角色	标注符号	含 义	例句（黑斜体字为标注的论元）
	系事	Arg1	参与者是联系动词联接的对象。 （系事是性状的系属者，是性状动词所描写的对象。与施事相比，系事不具有施动能力。）	你们 是 大学生。 他成为一代优秀学人。 这种款式很流行
	与事	Arg2	参与者是动作行为的间接承受者，也可用介词“给”、“为”、“对”等引进	小刘送我一本书。 老人把手艺传给了年青人。 他对新进的 设备 很爱惜
辅助 论元	方式	ArgM-MNR	指动作、行为进行的方式，可用介词“以（用）”引进。	他用草书写对联。 她用假声唱歌。 工人们用红砖砌墙
	方向	ArgM-DIR	指运行行为的方向，展示出沿路径的运动。“源”和“目标”都属于群组“direction”，另一方面，如果该运动路径没有“处所”标记，则应该使用	沿着这条路向东走一百米就到了。 从山上往下看景色美极了
	处所	ArgM-LOC	指动作行为发生的场所，可用介词“在（从、到）”引进。存现句的主语都是处所语义成分	墙上挂着一串串藤萝。 老板从墙上取下照片。 我们到北京再讨论吧
	时间	ArgM-TMP	指动作行为发生的时间，可用介词“在（从、到）”引进	我们于2009年考入大学。 三点钟我就在这儿等你了。 工作后他一直愁房子的事儿
	目的	ArgM- PNC	指动作、行为发生的目的，可用介词“为（为了）”引进	为了家人的健康，他不再吸烟了
	原因	ArgM- CAU	指动作行为发生的原因，可用“因为（由于）”引进	由于战争的失败，日本全面陷入美国的管制
	程度	ArgM-EXT	指动作行为产生的结果程度	鱼塘还没挖好。 作业还没写完
	情态	ArgM-MOD	指动作行为表示的情绪	我们应该原谅他。 我能不能不去上学
	否定	ArgM-NEG	指动作行为的否定状态	张三不喜欢李四。 不是所有的人都喜欢吃肉食
	连接	ArgM-DIS	表示一个句子连接到前面句子的标记	因此，他失去了上学的机会

表 10.10 以谓词“专注”框架为例，查看一下它在不同例句中的语义角色标注。

表 10.10 例句中的语义角色标注

Predicate	专注
Roleset ID	专注.01, focus, focus, concentrate Source: [focus.01]
Example	随着 金融 全球化，监测 应当 专注 于 整个 全球 金融 体系 的 稳定。 Arg0: 监测 ArgM-MOD: 应当 Rel: 专注 Arg2: 于 整个 全球 金融 体系 的 稳定 ArgM- CAU: 随着 金融 全球化
Example	科索沃 特派团 在 米特罗维察 继续 专注 于 推动 族群 间 的 活动 和 对话。 Arg0: 科索沃 特派团 ArgM-LOC: 在 米特罗维察 Rel: 专注 Arg2: 于 推动 族群 间 的 活动 和 对话
Example	这些 部分 也 专注 于 传播 信息 和 宣传 价值观。 Arg0: 这些 部分 ArgM-DIS: 也 Rel: 专注 Arg2: 于 传播 信息 和 宣传 价值观

10.2.2 PropBank 简介

良好的语义角色标注离不开语料资源的支持。英语较为知名的语义角色标注资源有 FrameNet、PropBank 和 NomBank 等。中文语义角色标注语料资源主要是从英语语义角色标注语料资源的基础上发展起来或参照其建设的。

中文命题库（Chinese Proposition Bank, CPB）来源于宾州树库（Chinese Penn TreeBank），它几乎对中文宾州树库中的每个动词及其语义角色进行了标注，国内大多数语义角色标注研究都是基于此资源。与英文 PropBank 基本类似，CPB 总共定义了 20 多个角色，只对每个句子中的核心动词进行了标注，所有动词的主要角色最多有 6 个，均以 Arg0~Arg5 和 ArgM 为标记。其中，核心的语义角色为 Arg0~Arg5 六种，其余为附加语义角色，用前缀 ArgM 表示，后面跟一些附加标记来表示这些参数的语义类别。

中文 PropBank (CPB) 项目的建设方法是在中文树库中加入手工解析的语义角色标注层。谓词及其对应特定论元的标注层被标注到依存树的成分中。句子中每个谓词的论元都以 ArgN 的方式标注, 这里 N 是 0~5 的整数。这些数字代表了谓词关系定义的核心论元, 标注为 Rel。每个核心论元都扮演着与谓词有关的重要角色, 通常数字的总数不超过 6。

与标准的语义角色标注一样, PropBank 也将论元分为核心论元和附加(辅助)论元。其标注符号与前文大致相同。PropBank 谓词论元文件被定义在一个 xml 文件中。下面以动词“便利”为例, 分析 Probank 对语义角色定义的细节。

(1) 通过 NLTK 或其他方式获得 cpb 3.0, 文件名为 LDC2013T13, 解压之后在 cpb3.0/docs 目录下找到 file.tbl 文件, 其中保存了所有 XML 文件的列表。

(2) 在 cpb3.0/data 目录中找到文件名: 00173-bian-li-v.xml。文件名分为三段。第一段为流水号, 从 00001 开始, 00173 是第 173 个词汇; 最后一段的“v”表示动词的词性, 名词为“n”; 中间段(可能有多段“-”)是词汇的汉语拼音。

10.2.3 CPB 中的特殊句式

PropBank 根据汉语语言的特点, 对标注内容制定了一些新的规范。

1. “把”字结构

在中文树库中, “把”被看作一个特殊的动词, 作为小句(IP)的补足语。CTB 中“把”的使用, 根据 IP 补足语中动词的情况, 显示出两类主要的模式。第一类模式, IP 补足语的动词可以是及物动词或双及物动词, 以及复合动词。“把”类动词一般可以转换为“非把”结构的陈述句式, 其含义是大致相同的。这种情况下, 两种变换的谓词论元结构是相同的。这意味着不需要考虑“把”有其自身的谓词论元结构, “把”的主语被认为是一个 IP 的补语论元。

警察将三名涉嫌犯罪分子逮捕。	REL: 逮捕
警察逮捕三名涉嫌犯罪分子。	Arg0: 警察
	Arg1: 三名涉嫌犯罪分子

第二类模式, CTB 中“把”的 IP 补足语的主动词被看作一个动词复合。

它不同于第一类模式, 有其他的论元跟随着复合动词, 在 CTB 中, 它被看作一个复合动词的宾语。“把”的使用存在如下几种情况:(1) “把”的主语仍被看作 IP 补足语的

主语。(2) IP 补足语的主语可被看作复合动词中第一个动词的宾语，而不是整句的宾语。(3) IP 补足语的宾语被复合动词的第二个动词引入。(4) “把”结构有时能有“非把”的转换，但不总是这样。

仔细观察，可以发现，(1) 中复合动词的第二动词的意义一般上都被“淡化”，而(2) 中的动词属于一个封闭的类。在这个意义上，第二个动词更像是介词而非动词。复合动词的整体行为更像是一个短语动词，其中第二个动词（或介词）引入了附加的论元。

苏联把东德建成对付西方的桥头堡。	REL: 建成 Arg0: 苏联
苏联建东德成对付西方的桥头堡。	Arg1: 东德 Arg2: 对付西方的桥头堡

2. “被”字结构

像“把”一样，“被”是一个轻动词（Light Verb，轻动词结构，真正的谓语通常是轻动词支持的名词化的谓词结构），它没有自己的谓词论元结构，也像“把”、“被”的存在会导致其 VP、IP 或 CP 的补足语中，动词的论元出现系统性错位。在一般情况下，VP/IP/CP 补足语动词必须是及物的、双及物的或以其他方式允许宾语前置。动词的主语可以缺失，在短“被”（标记为 SB）的情况下，“被”选取 VP 作为其补足语，否则就是一个长“被”（标记为 LB）。长“被”结构需要一个 IP 或 CP。

1) 长“被”结构

长“被”（词性标注为 LB）需要一个 IP 或 CP 作为其补足语，并且 IP 或 CP 的主语都不能缺少。它采用的 CP 补足语，当在 CP 补足语中动词的宾语错位时，在 CTB 中被明确表示为一个联接 CP 的共引用操作符。该操作符是对“被”主语的共指，虽然这并没有明确表示。当标注 CP 补足语的动词的谓词论元结构时，空的宾语将被替换为长“被”的主语，通过共索引的空操作符共指它。如下展示了双及物动词的谓词论元结构的标注，以及一个动词控制下的宾语。

今年，他被杭州大学录取。	REL: 录取 Arg0: 杭州大学 Arg1: 他 Arg2: 对付西方的桥头堡 ArgM-TMP: 今年
--------------	--

长“被”结构的另一个使用方法是，当一个短语动词出现在“被”的 IP 补足语时。在这种情况下，在 CTB 中没有明确地表示出错位。一般来说，长“被”的主语是一个短语动词/复合动词的中心词（第一个动词）的宾语。短语动词的目的是通过第二个动词（或介词）推出一个附加论元。如下展示了短语动词的谓语论元结构的标注。

植被被大车压成了路面。	REL: 压成
	ASP: 了
	Arg0: 大车
	Arg1: 植被
	Arg2: 路面
	ArgM-TMP: 今年

2) 短“被”结构

短“被”在很多方面与长“被”结构相似。最显著的区别是，补语从句（注释为一个 VP）的主语是缺失的。

费孝通被授予麦格赛赛奖。	REL: 授予
	Arg1: 麦格赛赛奖
	Arg2: 费孝通

也像长“被”结构一样，短“被”可与短语动词共现。短语动词/复合动词的类型与“把”字结构和长“被”结构中共现的类型相似。在这种情况下，在 CTB 中不明确地假定出空类，即使短“被”结构的主语就是短语动词的中心词的宾语。第二个动词（介词）引入了一种附加的论元，该论元在 CTB 中被标注为宾语。短语动词的谓词论元结构的标注类似于长“被”结构，只是缺失了 Arg0。而带有短“被”结构的共现词则使用其他的论元：列入、列为、确认为、用于、确定为、誉为、调往、称为、派往、推举为、聘请为、解职、任命为、推迟为、介绍到、选为、认定为、制定为、命名为、用于、视为、看作、审定为、应用到、称作、认定、纳入、封为。

住房却被当作职工福利。	REL: 当作
	Arg1: 住房
	Arg2: 职工福利

3. “由”字结构

对于某些及物动词有可能要前移宾语到主语位置之前，那么主语将通过介词“由”引入。这种结构通常具有一个“非由”的替代方案。

中国对外贸易经济合作部和国家统计局联合组织这次评选。	REL: 组织 Arg0: 中国对外贸易经济合作部和国家统计局
这次评选由中国对外贸易经济合作部和国家统计局联合组织。	Arg1: 这次评选 ArgM-MNR: 联合

10.2.4 名词性谓词的语义角色

虽然通用的标注框架对于动词和其名词化形式都是相同的，但对于特定名词化的谓词也存在一些问题。在中文中某些名词都不是真正意义的名词 (Nominalizations)，因为存在与之共享相同形态的动词。此外，并非所有名词化的谓词修饰成分都可被视为其论元或辅助成分。某些修饰成分只出现在谓词的名词形式，但从未出现在其动词形式。在这种情况下，不认为它们是谓词的论元。而且，某个词的动词和名词的论元实现方式也非常不同。一个名词化谓词的论元常常实现为一个名词短语。然而，当谓词中出现支持动词时，它们可能会出现名词短语之外。

当一个名词能有其名词化的谓词形式时，我们所感兴趣的是谓词的核心论元和辅助论元都共现在其名词形式和动词形式中。做出这个判断的标准是语义而不是语法。例如，一个作为动词辅助论元的副词，当它修饰这个名词化的谓词时，几乎总是实现为形容词，但它仍被认为是一种辅助论元，即使它的句法类别已经改变。例如，“充分”是一个名词化谓词“调查”的辅助论元 (ArgM-MNR)，因为它也被用于作为其动词形式对应的状语。还要注意，当谓词是一个动词时，Arg1 是一个名词短语，而当该谓词名词化时，它是一个介词短语。例如：

联合国工发组织近期充分调查了图们江地区工业项目现状。	Rel: 调查 ArgM-MNR: 充分 Arg1: 图们江地区工业项目现状
联合国工发组织近期对图们江地区的工业项目现状进行了充分调查。	Rel: 调查 ArgM-MNR: 充分 Arg1: 对图们江地区工业项目现状

在中文中，由于缺乏形态变化，形容词和副词经常有着相同的形式，只是通过谓词的名词和动词形式做出区分。而最可靠的区别是它们的句法分布。

1. NP 内部的论元

(1) 谓词是 NP 中心词。

在许多情况下，一个名词化谓词的所有论元可以位于 NP 内部，即 NP 的中心词。在

语法上，核心论元实现为谓词的修饰成分。在中文中，除极少数例外，这些修饰成分位于中心词的左边。根据它们到谓词的语义关系，其要么被标记为一个核心论元（ArgN），要么为辅助论元（ArgM）。名词性谓词的核心/辅助论元区分与动词的核心论元和辅助论元划分方法相一致：论元是由谓词确定的，因此必须遵循谓词的论元的选择限制。

这一地区成为海峡两岸科技、经贸合作的最佳地带。	Rel: 合作 Arg0: 海峡两岸 Arg1: 科技、经贸
-------------------------	--------------------------------------

例如，“合作”的 Arg0，必须有多个参与方，而且必须是能够合作的实体。另外，辅助成分（ArgM）可以修饰更广范围的谓词。名词通常伴随形容词/副词转换：副词修饰动词的谓词，而形容词修饰名词化谓词，尽管这些修饰成分共享相同的语义内容。因此，使用相同的功能标签对动词性和名词性谓词的 ARGMs 分类，这独立于句法分类。

(2) 谓词是 NP 中心词的修饰成分。

存在这样的情况，其中名词化谓词是另一个名词的修饰成分，这个被修饰的名词是整个名词短语中心词。当这个短语的中心词也是一个名词化谓词时，从这些中心词中区分修饰成分的核心论元等于歧义消除的形式。例如：

这是浙江省制定的“九五”期间对外经贸发展规划纲要。	Rel: 发展 Arg1: 对外经贸 ArgM-TMP: “九五”期间 head: 规划
---------------------------	---

有时候，一个名词化谓词所修饰的 NP 的中心词是名词化谓词的论元。这发生在谓词的功能作为一个简化关系从句的情况下。例如：

调查对象为北京城区十四至三十五岁的青年。	Rel: 调查 Arg1: 对象
----------------------	---------------------

2. 谓词做主语

中文的一个特点是广泛使用话题结构。当一个名词化谓词占据主语位置时，话题常常是这个谓词的论元。例如：

近年来，中韩两国之间的经贸往来发展迅速。	Rel: 发展 Arg1: 中韩两国之间的经贸往来
----------------------	------------------------------

3. 带有支撑（轻）动词的谓词

通常情况下，名词化谓词常出现支撑动词（第 6 章中的轻动词）。一些支撑动词有很少或没有语义内容，而且只是履行句法功能。另一些支撑动词起到添加意义到名词化的谓词中的作用。例如：

日本还应中国有关部门之邀，对中国长春至珲春的铁路沿线地区综合开发进行调查。	Rel: 调查
	Sup: 进行
	Rel: 开发
	Arg1: 中国长春至珲春的铁路沿线地区
	ArgM-MNR: 综合

一些支撑动词是“被动导向的”，它们的存在伴随着论元的翻转。如下例中的“受”。

目前，中国经济更易受国际环境的影响。	Sup: 受
	Rel: 影响
	Arg0: 国际环境
	Arg1: 中国经济
	ArgM-MNR: 综合

根据定义，轻动词之所以作为支撑动词，它必须至少与名词化谓词共享一个核心论元时。是否名词化谓词也共享了支撑动词辅助论元修饰成分，这是一个更棘手的问题。其答案部分地取决于支撑动词的语义水平；部分地取决于该辅助论元的性质。一般而言，支持动词的语义内容越少，辅助论元就越有可能由名词化谓词限制。

这是另一个领域，需要做很多模棱两可的方案。我们的策略是标记属于名词化谓词的辅助论元，当它显然是被许可的名词，而不是轻动词时。始终标记该（编号）论元。例如：

中外在工业、农业、经贸、文教等方面进行多种合作。	Sup: 进行
	Rel: 合作
	Arg0: 中外
	ArgM-LOC: 在工业、农业、经贸、文教等方面

在下例中，很明显，状语由支撑动词“扩大”提供而不是由名词化的谓词“合作”提供。值得一提的是，这个特殊的支撑动词都有自己的语义内容。

通过进一步扩大对外合作。	Sup: 扩大
	Rel: 合作
	ArgM-DIR: 对外

一个动词在某些情况下是支撑动词，但在其他的情况下则不是，这是有可能的。当它不与名词化谓词共享一个论元时，这类动词不应该被标记为支撑动词，即使它可能在其他上下文中共享论元。

4. 谓词位于介词短语内部

当一个名词化谓词出现在 NP 中，而这个 NP 是一个介词短语的补足语时，它的论元大致可以在介词短语外找到。这种说法往往是主句的（逻辑）的主语。例如：

这份报告是由美国哈佛大学和加州大学的知名专家学者经过近6个月的研究后提出的。	Rel: 研究 Arg0: 由美国哈佛大学和加州大学的知名专家学者 ArgM-TMP: 近6个月
--	--

名词化谓词的论元是否可以在介词短语外发现，这取决于介词。有些类型的介词不允许名词化谓词的论元在介词短语外被发现。例如：

海关工作要为国家财政和经济建设作出更大的贡献。	Rel: 建设 Arg1: 国家财政和经济
-------------------------	--------------------------

上述并没有穷尽名词化谓词所有可能的情况，但对于典型的场景，名词化谓词和它们的论元都是可以找到的。

10.2.5 PropBank 展开

本节的内容主要是 PropBank 命题库（CPB 3.0 谓词论元库）的存储和解析。需要读者已有 CPB 的完整版，后面代码提供的是 3.0 版，文件名为 LDC2013T13，解压之后所有的数据文件都位于 cpb 3.0 目录下，共有三个项目：index.html、docs、data。前两个是说明性的文件，而所有的数据都位于 data 目录下。在 data 目录下也有三个项目：frames 目录中已经提取出谓词论元结构，前文已经讲解过。cpb3.0-nouns.txt 中是名词性谓词论元的映射文件。cpb3.0-verbs.txt 中是动词性谓词论元的映射文件。这两个文件的映射内容都来自宾州树库 Treebank 7.0 的相关文件。

文件 cpb3.0-nouns.txt 和 cpb3.0-verbs.txt 的文件结构相同，均由如下编码格式的记录构成。

```
chtb_0168.nw 5 27 gold 便利.01-----23:1-ARG0-PSR 24:2-ARG0-PSE 27:0-rel
```

PropBank 语义角色记录解析如表 10.11 所示。

表 10.11 PropBank语义角色记录解析

对应的ctb文件名 (ctb 7.0或以上)	cttb_0168.nw
Treebank文件中的 句子序号	5
句子中词汇的序号	27 例如, ((IP (NP-SBJ (NP-APP (NP-PN (NR 荷兰)) (ADJP (JJ 驻华)) (NP (NN 大使))) (NP-PN (NR 郝德扬) (NN 先生))) (VP (PP-LOC (P 在) (LCP (NP (ADJP (JJ 揭幕)) (ADJP (JJ 剪彩)) (NP (NN 仪式))) (LC 上))) (VP (VV 说) (PU,) (IP-OBJ (IP-SBJ (NP-SBJ (-NONE- *pro*)) (VP (ADVP (AD 之所以)) (VP (VV 选择) (IP-OBJ (NP-SBJ (-NONE- *PRO*)) (VP (PP-LOC (P 在) (NP-PN (NR 武汉))) (VP (VV 设立) (NP-OBJ (NN 办事处)))))) (PU,) (VP (VC 是) (PP-PRD (P 因为) (IP (NP-PN-TPC (NR 武汉)) (IP (IP (NP-SBJ (NP (NN 水) (NN 陆)) (NP (NN 交通))) (VP (VA 便利))) (PU,) (IP (NP-SBJ (NN 地理) (NN 位置)) (VP (VA 优越)))))) (PU。)))
谓词论元Id	便利.01, 01对应框架文件00173-bian-li-v.xml, frameset id = f1
标注方式	gold (经人工两次校对以上, 高精度)
分隔符	—
句子中谓词论元的位置	23:1-ARG0-PSR: 表示ARG0-PSR这个论元位于句法树的第23个叶子节点, 树高度为1。截取结果为: “武汉”, 截取的句法树分枝为: “Tree('NP-PN-TPC', [Tree('NR', ['武汉'])])” 24:2-ARG0-PSE: 表示ARG0-PSR这个论元位于句法树的第24个叶子节点, 树高度为2。截取结果为: “水 陆 交通”, 截取的句法树分枝为: “Tree('NP-SBJ', [Tree('NP', [Tree('NN', ['水']), Tree('NN', ['陆'])]), Tree('NP', [Tree('NN', ['交通'])])])” 27:0-rel: 表示谓词位于句法树的第27个叶子节点, 树高度为0。截取结果为: “便利”截取的句法树分枝为: “Tree('VA', ['便利']) ”

注：这里的句法树格式使用了 Python NLTK 的内部句法数据格式，用“repr”指令输出。该数据格式在还原句法树后，严格保持原有的字符集，不会出现诸如全角符号“，”变为半角符号“;”的情况。

所谓 PropBank 的展开，就是根据前面解析出的文件编码规则，将 Treebank 中对应的句法树分枝映射到 PropBank 中，这么做有助于对谓词论元结构进一步分析。

我们计划将 PropBank 解析后的编码和展开后的结果放到数据库中存储，首先需要定义一张数据表。


```
CREATE TABLE `propank` (  
  `Id` int(11) NOT NULL AUTO_INCREMENT,  
  `filename` varchar(255) DEFAULT NULL COMMENT '参考文件名',  
  `seq` int(11) DEFAULT NULL,  
  `treeid` int(11) DEFAULT NULL COMMENT '对应的树库 id',  
  `predicatePOS` varchar(50) DEFAULT NULL COMMENT '谓词词性',  
  `rel` varchar(255) DEFAULT NULL COMMENT '谓词',  
  `srlstr` text COMMENT '语义字符串',  
  `augments` text COMMENT '论元结构',  
  PRIMARY KEY (`Id`)  
) ENGINE=MyISAM AUTO_INCREMENT=187732 DEFAULT CHARSET=utf8 COMMENT='动词角色标注';
```

在第 7 章中，已经将 CPB 保存到数据库中，为了便于说明下面给出 PropBank 和 Treebank 字段对照表，如表 10.12 所示。

表 10.12 PropBank和Treebank字段对照表

PropBank字段	字段说明	CPB 8.0字段
Id	Id	—
filename	参考文件名	filename
seq	句子序列号	seq
treeid	对应的宾州树库的Id	Id
predicatePOS	谓词词性: noun、verb	—
	动词和名词分属两个不同的文件: cpb3.0-verbs.txt 和 cpb3.0-nouns.txt, 展开时, 根据文件指定即可	
rel	谓词: 便利.01	—
srlstr	语义编码字符串: chth_0168.nw 5 27 gold 便利.01 ---- 23:1-ARG0-PSR 24:2-ARG0-PSE 27:0-rel	—
augments	论元结构: 展开后的结果: rel: Tree('VA', ['便利']) ARG0-PSR: Tree('NP-PN-TPC', [Tree('NR', ['武汉'])]) ARG0-PSE: Tree('NP-SBJ', [Tree('NP', [Tree('NN', ['水']), Tree('NN', ['陆'])]), Tree('NP', [Tree('NN', ['交通'])])])	从flatTree中截取

展开的过程分为如下三个步骤完成。

(1)NLTK 中 read_instance_block 方法有最大记录的限制,手工增加一个 read_instance 方法,如图 10.3 所示。

```

158 | # customize API $
159 | def read_instance(self, stream, instance_filter=lambda inst: True):$
160 |     line = stream.strip()$
161 |     if line:$
162 |         inst = PropbankInstance.parse($
163 |             line, self._parse_fileid_xform,$
164 |             self._parse_corpus)$
165 |     return inst$
166 | #####
167 | #{ Propbank Instance & related datatypes$
168 | #####

```

图 10.3 手工增加一个 read_instance 方法

将该方法放到文件 nltk 的 X:\Anaconda\Lib\site-packages\nltk\corpus\reader 目录下的 propbank.py 文件中。

图 10.3 中的行号是放置的位置。

(2) 解析 PropBank 字符串。在 treelib 中，增加导入模块和函数代码如下。

```

import re
import nltk
from nltk.tree import *
from framework import *
from MySQLdb import *
from nltk.corpus import propbank

# 该函数用于解析 prop 中的语义编码字符串，并从 treebank 中找到对应的 treeid
def parseProp(propstr, postag, DBConn):
    proplist = []
    prefix = propstr.split("-----")
    term = prefix[0].split(" ")
    filename = term[0]
    seq = str(int(term[1])+1)
    predicatePOS = postag
    rel = term[4]
    srlstr = propstr
    sql = "SELECT * FROM treebank WHERE filename = '"+filename+"' AND seq = '"+seq+"'"
# 查询 treebank 找出对应的 treeid
    record = DBConn.query(sql)
    treeid = str(record[0]['Id'])
    flattree = record[0]['flatTree']
    augments = get_augments(flattree, srlstr, treeid)
    proplist.append(escape_string(filename)); proplist.append(seq);
    proplist.append(treeid); proplist.append(predicatePOS);
    proplist.append(escape_string(rel)); proplist.append(escape_string(srlstr));

```

```
proplist.append(escape_string(augments))
return proplist
```

(3) 映射论元结构, 获取 `augments` 字段信息。该函数位于 `treelib` 中。

```
def get_augments(treestr, srlstr, treeid): # 该函数用于从 tree 中截取句法树枝结构
    augments = ""
    tree = Tree.fromstring(treestr)
    try:
        inst = propbank.read_instance(srlstr)
        treepos = inst.predicate.treepos(tree)
        relstr = repr(tree[treepos]).replace("\n", "") + "\n"
        augments += "rel: " + relstr.decode("unicode-escape")
        auglist = [ str(argid) + ": " + repr(argloc.select(tree)).decode("unicode-escape").replace("\n", "") for (argloc, argid) in inst.arguments ]
        if auglist: augments += u"\n".join(auglist)
    except Exception, e:
        print e
        appendfile("emptytree.txt", srlstr + "-->treeid:" + str(treeid) + "\n")
        sys.exit()
    return augments
```

(4) 执行程序如下。

```
# -*- coding: utf-8 -*-
import sys, os
import nltk
from framework import *
from treelib import *
from MySQL import CMySQL
from MySQLdb import *
from nltk.tree import Tree

reload(sys) # 设置 UTF-8 输出环境
sys.setdefaultencoding('utf-8')
# 关联 verb 的 treeid
root = "X:\\CTB_Corpus\\treebanks\\cpb3.0\\data\\"
verbdir = root + "cpb3.0-verbs.txt"
verblist = readfile(verbdir).splitlines()
print len(verblist)
DBconn = CMySQL("127.0.0.1", "root", "root", "testbank", 3306)
# Id, filename, seq, treeid, predicatePOS, rel, srlstr, augments
for prop in verblist:
    proplist = parseProp(prop, "verb", DBconn)
    sql = "INSERT INTO propbank VALUES ('', '"+', "',''.join(proplist)+'')"
```



```
DBconn.execute(sql)

print "ok"
```

PropBank 解析后的结果如图 10.4 所示。

1	chtb_0001.nw	1	1	verb	同步 01	<MEMO>	<MEMO>
2	chtb_0001.nw	11	11	verb	完 01	<MEMO>	<MEMO>
3	chtb_0001.nw	3	3	verb	确保 01	<MEMO>	<MEMO>
4	chtb_0001.nw	3	3	verb	颁布 01	<MEMO>	<MEMO>
5	chtb_0001.nw	3	3	verb	实行 01	<MEMO>	<MEMO>

文本 十六进制编辑器

Tree(VV, [实行])ARGO, Tree(NP-PN-SBJ, [Tree(NR, [上海]), Tree(NR, [浦东])])ARGM-TMP, Tree(LCP-TMP, [Tree(NP, [Tree(NT, [近年])]), Tree(LC, [来])])ARG1, Tree(NP-OBJ, [Tree(CP, [Tree(WH-NP-I, [Tree(NONE, [“OP”])]), Tree(CP, [Tree(IP, [Tree(NP-SBJ, [Tree(NONE, [“I”])]), Tree(VP, [Tree(VV, [涉及]), Tree(NP-OBJ, [Tree(NP-APP, [Tree(NN, [经济]), Tree(PU, [、]), Tree(NN, [贸易]), Tree(PU, [、]), Tree(NN, [建设]), Tree(PU, [、]), Tree(NN, [规划]), Tree(PU, [、]), Tree(NN, [科技]), Tree(PU, [、]), Tree(NN, [文教]), Tree(ETC, [等])]), Tree(NP, [Tree(NN, [领域])])]), Tree(DEC, [B?])]), Tree(QP, [Tree(CD, [七十一]), Tree(CLP, [Tree(M, [件])]), Tree(NP, [Tree(NN, [法规性]), Tree(NN, [文件])])])])])

图 10.4 PropBank 解析后的结果

由于这里使用的是 CTB 8.0，有一小部分的语义论元编码字符串与 cpb 3.0 中的记录不符，但数量不多，所以我们在代码中加入了异常处理，异常信息会被记录到 emptytree.txt 文件中。通过核对这个文件的记录，可以对错误做出修正。

10.3 句子的语义解析

语义角色标注给出了句子语义解析的理论来源，而语义依存算法则是一个整合了语义角色标注和句法分析的合成实现模型。它的理论基础可以追溯到第 2 章的三个平面理论，以及语言学家所公认的一些经验事实之中：“凡是由实词和实词构成的汉语句子中，一定同时并存着两种结构关系，它们分别是句法关系和语义关系。而这两种结构关系间并不是一一对应的”，多年的研究使这一论断具有完整而坚实的理论基础。

10.3.1 语义依存

语义依存分析是哈工大社会计算与信息检索研究中心最新研究的一种基于语义依存关系的自动句义分析系统。所谓语义依存是指在句子结构中分析实词和实词之间的语义关系，这种关系是一种基于事实的概念依存关系，只有当词语进入到句子时才会存在。语义依存分析目标是跨越句子表层句法结构的束缚，直接获取深层的语义信息。

LTP-cloud 网站上发布了一个语义依存的 DEMO 版本（URL：<http://www.ltp-cloud.com/demo/>），为了进一步说明，下面给出几个例子，如表 10.13 所示。

表 10.13 几个语义依存的例子

<p>北京大学录取了这名考生</p> <p>语义角色标注:</p> <p>Rel: 录取</p> <p>Agt(Arg0): 北京大学</p> <p>Pat(Arg1): 这名考生</p>	<p>Root 北京大学 录取 了 这 名 考生</p> <p>ni v u r q n</p> <p>机构</p> <p>A0 录取 A1</p>
<p>北京大学把这名考生录取了</p> <p>语义角色标注:</p> <p>Rel: 录取</p> <p>Agt(Arg0): 北京大学</p> <p>Pat(Arg1): 这名考生</p>	<p>Root 北京大学 把 这 名 考生 录取 了</p> <p>ni p r q n v u</p> <p>机构</p> <p>A0 A1 录取</p>
<p>这名考生被北京大学录取了</p> <p>语义角色标注:</p> <p>Rel: 录取</p> <p>Agt(Arg0): 北京大学</p> <p>Pat(Arg1): 这名考生</p>	<p>Root 这 名 考生 被 北京大学 录取 了</p> <p>r q n p ni v u</p> <p>机构</p> <p>A1 A0 录取</p>

表 10.13 中的三个句子，其句法结构各不相同，但是表达了同一个语义信息，那么其语义依存解析的结果也相同。

以往的 NLP 任务都是以句法分析作为核心的分析流程，句法解析是通向语义角色标注等后续阶段的必经之路。虽然有人也提出过从语义组块直接映射到语义角色标注的方

法，但是由于精度问题，始终没有获得本质的突破。语义依存方法的提出则从根本上解决了语义分析问题，依存原理来源于配价语法，而配价语法是基于语义的一种支配与从属关系，因此依存的概念本质上也是一种基于语义的关系。句法依存仅仅是借用了依存的特征来实现句法的解析。

语义角色标注是公认的能够真正解析句义数据结构，通过对句子的施事、受事、与事等核心角色，以及地点、时间、行为方式等附属角色的分析，语义角色标注天然地与词汇的配价理论相一致，从理论上当然能够转化为依存结构。

以往的语义角色标注的自动分析需要以句法分析为基础，句法分析的错误会带入到语义角色标注中。而语义依存分析直接在基本语言处理的基础上一步走到比语义角色标注更深层的语义分析阶段。这个过程在一步中完成将极大减少错误的级联，大大提高了句子语义角色标注的精度和速度。

语义依存分析的系统架构如图 10.5 所示。

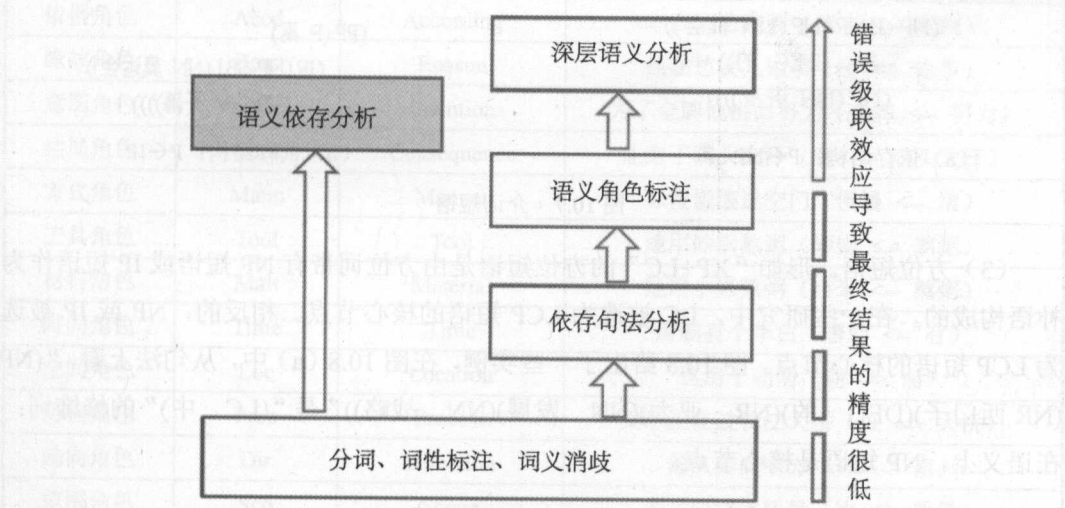


图 10.5 语义依存分析的系统架构

关于语义依存标注规范，哈工大给出了如下详细的说明。

语义依存与句法依存所使用的依存结构建立规则是不同的，语义依存分析直接反映语义信息，因此具有语义关系的词对直接构成弧，而不需要经过介词或助词。这些不同，比较明显地体现在 DNP/DVP 短语、介词短语、方位短语中。

(1) 并列结构。并列结构包括词级别、短语级别、从句级别的并列。不论是对于哪一个级别，依存结构建立都按照“最后一个并列成分作为核心节点”的原则进行。在并列结构中，最后一个单词、短语或从句被选为核心节点。如图 10.6 (a) 所示，(NN 和平)、

(CC 与)和(NN 安全)被语法分析为姐妹节点。在语义依存中,如图 10.6 (b) 所示,最后一个构成成分(NN 安全)被标记为核心节点,(NN 和平)和并列连词(CC 与)被标记为修饰节点。

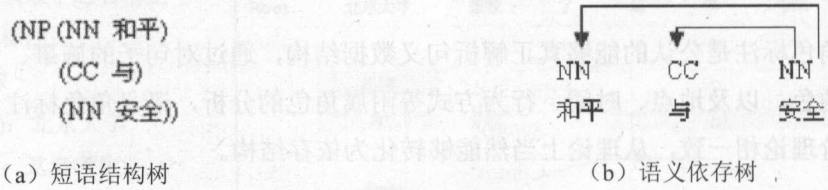


图 10.6 并列结构

(2) 介词短语。在句法上,介词短语中的介词被选为核心节点。为了更贴近语义,在语义依存中,将介词短语的宾语作为核心节点。图 10.5 给出一些实例,在图 10.7 (b) 中,从句法上看,核心节点是“P(离)”;在语义上,IP 为核心节点。

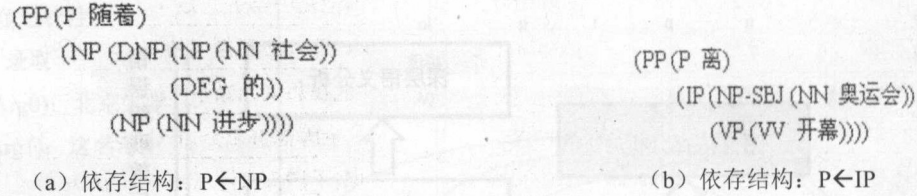


图 10.7 介词短语

(3) 方位短语。形如“XP+LC”的方位短语是由方位词带有 NP 短语或 IP 短语作为补语构成的。在一些研究中,LC 被选为 LCP 短语的核心节点。相反的, NP 或 IP 被选为 LCP 短语的核心节点。图 10.8 给出了一些实例,在图 10.8 (a) 中,从句法上看,“(NP (NR 西门子)(DEG 的)(NR 亚太)(NN 发展)(NN 战略))”是“(LC 中)”的修饰词;在语义上, NP 短语是核心节点。

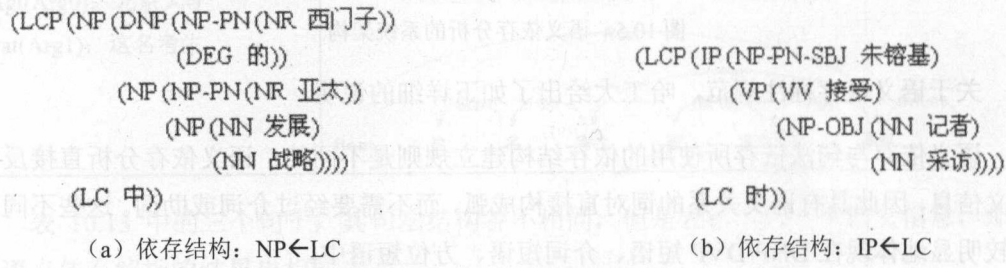


图 10.8 方位短语

下面给出语义依存分析的标注规范 (URL: http://www.ltp-cloud.com/intro/#sdp_how), 如表 10.14 所示。

表 10.14 LTP语义依存分析的标注规范

关系类型	Tag	Description	Example
施事关系	Agt	Agent	我送她一束花（我 <-- 送）
当事关系	Exp	Experiencer	我跑得快（跑 --> 我）
感事关系	Aft	Affection	我思念家乡（思念 --> 我）
领事关系	Poss	Possessor	他有一本好书（他 <-- 有）
受事关系	Pat	Patient	他打了小明（打 --> 小明）
客事关系	Cont	Content	他听到鞭炮声（听 --> 鞭炮声）
成事关系	Prod	Product	他写了一本小说（写 --> 小说）
源事关系	Orig	Origin	我军缴获敌人四辆坦克（缴获 --> 坦克）
涉事关系	Datv	Dative	他告诉我一个秘密（告诉 --> 我）
比较角色	Comp	Comitative	他成绩比我好（他 --> 我）
属事角色	Belg	Belongings	老赵有俩女儿（老赵 <-- 有）
类事角色	Clas	Classification	他是中学生（是 --> 中学生）
依据角色	Accd	According	本庭依法宣判（依法 <-- 宣判）
缘故角色	Reas	Reason	他在愁女儿婚事（愁 --> 婚事）
意图角色	Int	Intention	为了金牌他拼命努力（金牌 <-- 努力）
结局角色	Cons	Consequence	他跑了满头大汗（跑 --> 满头大汗）
方式角色	Mann	Manner	球慢慢滚进空门（慢慢 <-- 滚）
工具角色	Tool	Tool	她用砂锅熬粥（砂锅 <-- 熬粥）
材料角色	Malt	Material	她用小米熬粥（小米 <-- 熬粥）
时间角色	Time	Time	唐朝有个李白（唐朝 <-- 有）
空间角色	Loc	Location	这房子朝南（朝 --> 南）
历程角色	Proc	Process	火车正在过长江大桥（过 --> 大桥）
趋向角色	Dir	Direction	部队奔向南方（奔 --> 南）
范围角色	Sco	Scope	产品应该比质量（比 --> 质量）
数量角色	Quan	Quantity	一年有365天（有 --> 天）
数量数组	Qp	Quantity-phrase	三本书（三 --> 本）
频率角色	Freq	Frequency	他每天看书（每天 <-- 看）
顺序角色	Seq	Sequence	他跑第一（跑 --> 第一）
描写角色	Desc(Feat)	Description	他长得胖（长 --> 胖）
宿主角色	Host	Host	住房面积（住房 <-- 面积）
名字修饰角色	Nmod	Name-modifier	果戈里大街（果戈里 <-- 大街）
时间修饰角色	Tmod	Time-modifier	星期一上午（星期一 <-- 上午）
反角色	r + main role		打篮球的小姑娘（打篮球 <-- 姑娘）

续表

关系类型	Tag	Description	Example
嵌套角色	d + main role		爷爷看见孙子在跑 (看见 --> 跑)
并列关系	eCoo	event Coordination	我喜欢唱歌和跳舞 (唱歌 --> 跳舞)
选择关系	eSelt	event Selection	您是喝茶还是喝咖啡 (茶 --> 咖啡)
等同关系	eEqu	event Equivalent	他们三个人一起走 (他们 --> 三个人)
先行关系	ePrec	event Precedent	首先, 先
顺承关系	eSucc	event Successor	随后, 然后
递进关系	eProg	event Progression	况且, 并且
转折关系	eAdvt	event adversative	却, 然而
因果关系	eCau	event Cause	因为, 既然
结果关系	eResu	event Result	因此, 以致
推论关系	eInf	event Inference	才, 则
条件关系	eCond	event Condition	只要, 除非
假设关系	eSupp	event Supposition	如果, 要是
让步关系	eConc	event Concession	纵使, 哪怕
手段关系	eMetd	event Method	—
目的关系	ePurp	event Purpose	为了, 以便
割舍关系	eAban	event Abandonment	与其, 也不
选取关系	ePref	event Preference	不如, 宁愿
总括关系	eSum	event Summary	总而言之
分叙关系	eRect	event Recount	例如, 比方说
连词标记	mConj	Recount Marker	和, 或
的字标记	mAux	Auxiliary	的, 地, 得
介词标记	mPrep	Preposition	把, 被
语气标记	mTone	Tone	吗, 呢
时间标记	mTime	Time	才, 曾经
范围标记	mRang	Range	都, 到处
程度标记	mDegr	Degree	很, 稍微
频率标记	mFreq	Frequency Marker	再, 常常
趋向标记	mDir	Direction Marker	上去, 下来
插入语标记	mPars	Parenthesis Marker	总的来说, 众所周知
否定标记	mNeg	Negation Marker	不, 没, 未
情态标记	mMod	Modal Marker	幸亏, 会, 能
标点标记	mPunc	Punctuation Marker	, . !
重复标记	mPept	Repetition Marker	走啊走 (走 --> 走)

续表

关系类型	Tag	Description	Example
多数标记	mMaj	Majority Marker	们, 等
实词虚化标记	mVain	Vain Marker	
离合标记	mSepa	Seperation Marker	吃了个饭(吃 --> 饭)洗了个澡(洗 --> 澡)
根节点	Root	Root	全句核心节点

最后, 简单谈谈语义依存分析的算法及训练数据。语义依存分析可以有两种算法思路: 一种是基于图的方法; 另一种是基于深度学习的方法。《汉语语义依存分析》一文给出了基于图的算法, 基于图的方法需要分别计算句子的语义依存结构和依存弧上关系的权重, 搜索算法使用 Eisner 算法, 弧权重使用 Online 算法。

算法使用的特征由两部分构成: 一部分由依存弧本身构成的特征; 另一部分为语义关系构成的特征。其中, 弧特征用来计算两个节点之间构成一条弧的概率; 关系特征计算了节点 K 作为儿子节点或者核心节点, 依存弧方向为 dir 时, 对应依存关系为 t 的概率。这两种特征选择的方式, 都是仅考虑了两个有依存关系的词之间的信息。为了得到更精确的边权重信息, 还使用更高阶的特征、每个词的子孙节点及兄弟节点的信息。

(1) Eisner 算法。该算法以 span 为解码的基本单位, span 表示输入句子的一个片段对应的子树。与组块不同的是, span 中的核心词只能位于片段首或尾, 即 span 只包括这个词左边或者右边的子孙节点。另外, 除核心词外的另外一个片段首或尾词的修饰成分可以是不完整的, 即 span 可以不包括这个词左边的子孙节点或者右边的子孙节点。对于其他词, span 包括其所有的子孙节点。span 的这种特性使得解码算法独立地确定一个词左边的修饰成分和右边的修饰成分, 从而降低算法的复杂度。

(2) Online 算法。该算法是一种基于距离最大化的学习算法, 在文本分类、序列标注、依存分析问题上都表现出了很好的性能。本文采用的是 MST—Parser 中的方法, 即 K—MIRA(KBestMargin Infused Relaxed Algorithm)。

本文采用 4 000 句进行训练、620 句进行测试。表 10.15 给出了实验结果。其中, UAS 表示依存弧准确率(不含依存关系), LAS 表示依存关系准确率。

表 10.15 LTP语义依存的算法精度

算 法	UAS	LAS
基于图的算法	80.21%	65.62%
基于图的算法+最大熵模型	81.36%	68.18%

——《汉语语义依存分析》(郭 江, 车万翔, 刘 挺)

《汉语语义依存分析》一文发表于 2011 年，距今已经有 5 年时间，而语义依存模块发布时间是 2016 年，估计算法应该有所改变，精度也较之以往有很大提高。由于目前该项目没有开源，也没有详尽的资料，所以尚不得而知。不过所幸的是，哈工大发布了语义依存的 CoNLL 格式的标注训练和测试数据，中文语义依存分析可在两个语料库上进行评测。其中，THU 文件夹内为清华大学语义依存网络语料，HIT 文件夹内为哈尔滨工业大学依存语料库。如下展示数据是基于宾州树库（CTB）的，分词和词性标注之后两列加入了依存弧和语义标签，读者可以根据此数据训练自己的算法。

1	建筑	建筑	NN	NN	_	2	r-agent	_	_
2	公司	公司	NN	NN	_	3	agent	_	_
3	进	进	VV	VV	_	15	s-succession	_	_
4	区	区	NN	NN	_	3	LocationFin	_	_
5	,	,	PU	PU	_	15	PU	_	_
6	有关	有关	JJ	JJ	_	7	d-restrictive	_	_
7	部门	部门	NN	NN	_	9	agent	_	_
8	先	先	AD	AD	_	9	TimeAdv	_	_
9	送上	送上	VV	VV	_	15	s-succession	_	_
10	这些	这些	DT	DT	_	12	d-deno	_	_
11	法规性	法规性	NN	NN	_	12	d-attribute	_	_
12	文件	文件	NN	NN	_	9	possession	_	_
13	,	,	PU	PU	_	15	PU	_	_
14	然后	然后	AD	AD	_	15	TimeAdv	_	_
15	有	有	VE	VE	_	0	ROOT	_	_
16	专门	专门	JJ	JJ	_	17	d-attribute	_	_
17	队伍	队伍	NN	NN	_	15	content	_	_
18	进行	进行	VV	VV	_	15	content	_	_
19	监督	监督	NN	NN	_	20	s-coordinate	_	_
20	检查	检查	NN	NN	_	18	content	_	_
21	。	。	PU	PU	_	15	PU	_	_

训练数据可从数据堂免费下载（URL:<http://www.datatang.com/data/44116>）。

虽然目前有可能尚不存在针对语义依存分析的实用算法，但语义依存概念的提出和在此基础上建立的语义标注规范是对语义解析的一种新突破，对更高层次的研究和应用，如自动问答、信息抽取、机器翻译、信息检索、自动文摘等具有深远的意义。

10.3.2 完整架构

根据上述分析给出句子的语义解析架构，如图 10.9 所示。

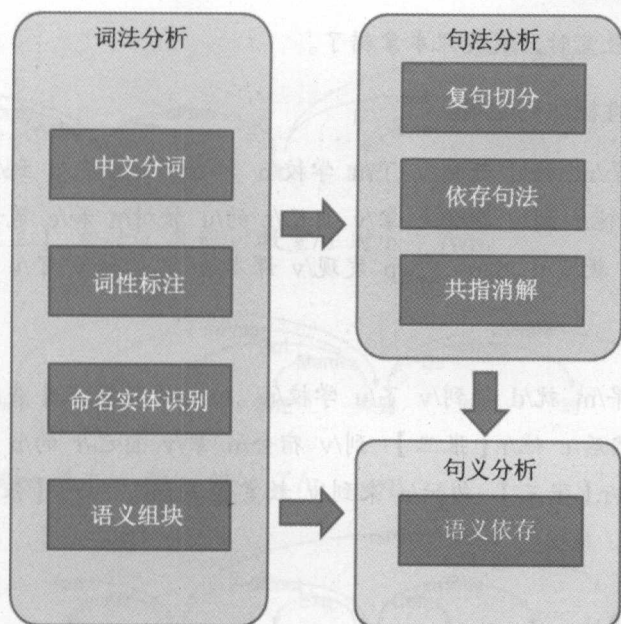


图 10.9 句子的语义解析架构

该架构共包含三大部分，分别为词法分析、句法分析和句义分析。

第一步为词法分析，该阶段有两个任务：一个是词形的解析，包括：中文分词、命名实体识别、语义组块；另一个是词性的解析，包括：词性标注和组块标注。

词法分析完成之后，其结果可以进入第二步句法分析。在句法分析阶段，首先要对复句进行切分，变为单句集合的形式。然后进行依存句法（或短语结构句法）分析，获得分析结果进入共指消解，还原被压缩部分的指称成分。

第三步为句义分析，分为如下几种情况。

- ☐ 如果输入的是单句，也不需要共指消解，那么直接使分词、词性标注后的句子进入第三步进行句义分析。
- ☐ 如果输入的是复句，但不需要共指消解，那么将分词、词性标注后的句子先做复句切分，切分之后的结果再做句义分析。
- ☐ 如果输入的是复句，而且需要共指消解，那么将分词、词性标注后的句子先做共指消解，再做复句切分，最后将结果做句义分析。

最后，用如下例子来解释整个过程。

(1) 例句。

张三大早就赶到了学校。他先到食堂吃早餐，然后他到宿舍拿自己的教材和笔记

本。当他匆忙来到教室时，发现课本拿错了。

(2) 分词、词性标注后结果。

张三/nh 一大早/nt 就/d 赶到/v 了/u 学校/n 。/wp 他/r 先/d 到/p 食堂/n 吃/v 早餐/n ，/wp 然后/c 他/r 到/v 宿舍/n 拿/v 自己/r 的/u 教材/n 和/c 笔记本/n 。/wp 当/p 他/r 匆忙/a 来到/v 教室/n 时/n ，/wp 发现/v 课本/n 拿/v 错/v 了/u 。/wp

(3) 共指消解。

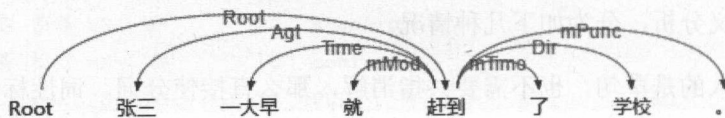
张三/nh 一大早/nt 就/d 赶到/v 了/u 学校/n 。/wp 他/r【张三】先/d 到/p 食堂/n 吃/v 早餐/n ，/wp 然后/c 他/r【张三】到/v 宿舍/n 拿/v 自己/r 的/u 教材/n 和/c 笔记本/n 。/wp 当/p 他/r【张三】匆忙/a 来到/v 教室/n 时/n ，/wp 【张三】发现/v 课本/n 拿/v 错/v 了/u 。/wp

(4) 复句切分。

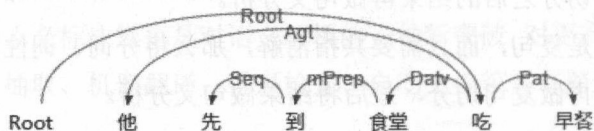
- 张三/nh 一大早/nt 就/d 赶到/v 了/u 学校/n 。/wp
- 他/r【张三】先/d 到/p 食堂/n 吃/v 早餐/n ，/wp
- 然后/c 他/r【张三】到/v 宿舍/n 拿/v 自己/r 的/u 教材/n 和/c 笔记本/n 。/wp
- 当/p 他/r【张三】匆忙/a 来到/v 教室/n 时/n ，/wp
- 【张三】发现/v 课本/n 拿/v 错/v 了/u 。/wp

(5) 语义依存。

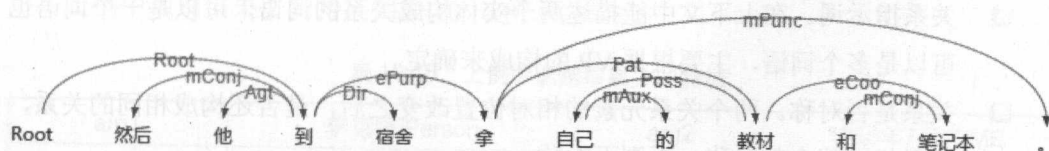
张三/nh 一大早/nt 就/d 赶到/v 了/u 学校/n 。/wp



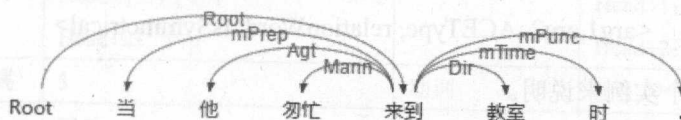
他/r【张三】先/d 到/p 食堂/n 吃/v 早餐/n ，/wp



然后/c 他/r【张三】到/v 宿舍/n 拿/v 自己/r 的/u 教材/n 和/c 笔记本/n 。/wp



当/p 他/r【张三】 匆忙/a 来到/v 教室/n 时/n , /wp



【张三】 发现/v 课本/n 拿/v 错/v 了/u 。 /wp



通过该架构可以完整地解析一个句子的语义内容，并可用于相应的诸如问答系统、框架语义抽取、简单对话这类更高级的 NLP 应用系统之中。

10.3.3 实体关系抽取

语义依存使 NLP 走向大规模应用成为可能。最新的 NLP 应用，诸如问答系统中的问句分析、问句检索、答案抽取都离不开语义依存。除此之外，情感分析中的语义倾向判断、评价对象抽取；信息抽取中的事件抽取、主题抽取等；信息分类的文本分类、语言意图分类，以及句子相似度计算等能看到语义角色标注的身影。

本文的主要内容大多涉及一些 NLP 的基础层应用，结合前文悬而未决的知识库实体关系抽取问题，这里给出使用语义角色标注来解决此类问题的一些尝试。实体关系抽取是自动构建知识库的最重要的领域。这方面的研究很多，常用的实体关系与 RDF 和 OWL 的规则是一致的，基本形式已经比较稳定。下面通过一个实例来看一下实体关系抽取的规范。目前实体关系被描述为一个五元组。

- 实体元素 1。构成关系的第一个实体。
- 实体元素 2。构成关系的第二个实体。
- 实体关系类型。实体之间构成的关系类型可用 ACE 实体关系类型或 HowNet 的语义角色来描述。

- ❑ 关系指示词。在上下文中能描述两个实体构成关系的词语，可以是一个词语也可以是多个词语，主要根据 VP 的构成来确定。
- ❑ 关系是否对称。两个关系元素的相对位置改变之后，是否还构成相同的关系，如果相同那么就对称，否则不对称。

该五元组可以表示为如下形式。

`<arg1,arg2, ACEType, relationWord, isSymmetrical>`

下面通过一个实例来说明。

例句：国务院/Organization 总理/n 李克强/Person 出生/v 于/p 1955 年 7 月/TIME。

识别结果 1：<国务院，李克强，机构关系，总理，否>

识别结果 2：<李克强，1955 年 7 月，出生，否>

目前实体关系抽取常用的方法大致有如下两种。

1) 基于规则的方法

通过人工书写来描述两个实体所在结构的规则，前文中的 DBPedia 主要采取这种方法。这种方法要求规则构建者对领域的特点有深入的了解，投入成本大、移植性差，所以逐渐被其他方法所取代。前面已经详细介绍了，这里不再赘述。

2) 基于特征的方法

本节所选择的特征分为两类：一类是一般常用的特征；另一类是语义角色特征。根据特征训练一个分类器，之后使用训练好的分类器完成关系抽取任务。经过多年的研究，常用分类器有最大熵分类器和支持向量机，并且支持向量机的效果最好。由于机器学习算法已经发展得比较成熟，因此实体关系抽取转化为分类问题后，其关键问题就在于有效特征的选择。

(1) 常用特征。所谓常用特征，主要是指以实体的上下文、词性、动词特征、距离特征，以及使用句法信息作为抽取特征。常用特征集如下。

- ❑ 实体及其上下文特征主要包含实体中心词、实体前两个词、实体后两个词，以及这些词的词干和词性。实体及其上下文特征是最基本、最简单的特征。
- ❑ 动词特征：句子中的所有动词。
- ❑ 距离特征：指要抽取实体关系的两个实体间的词距。
- ❑ 实体扩展特征：指实体的同义词和上位词。

上例中识别结果 2 的特征如表 10.16 所示。

表 10.16 上例中识别结果 2 的特征

arg1	李克强/Person	arg2	1955年7月/TIME
arg1的上下文	Head-2: 国务院/Organization, Head-1: 总理/n Head+1: 出生/v Head+2: 于/p	arg2的上下文	Head-2: 出生/v Head-1: 于/p Head+1: NULL Head+2: NULL
arg1与arg2的距离	3	动词	出生/V
实体扩展特征	略	—	—

(2) 语义角色特征。所谓语义角色特征就是句子的语义角色标注或语义依存标注。在使用之前, 需要为语义角色特征做出如下定义。

- 语义角色对 (Semantic Role Pair, SRP) 特征定义: 若实体 e_1 相对于谓词 V 的语义角色为 R_1 , 实体 e_2 相对于 V 的语义角色为 R_2 , 则 (R_1, R_2) 称为实体对 (e_1, e_2) 相对于谓词 V 的语义角色对特征。实体对 (e_1, e_2) 在不同的谓词框架中将构成不同的语义角色对特征。
- 谓词 (Predicate, Pred) 特征定义: 若实体对 (e_1, e_2) 相对于谓词 V 存在语义角色对 (R_1, R_2) , 则 V 的词干 $s(V)$ 称为实体对 (e_1, e_2) 的谓词特征。
- 语义角色对+谓词组合 (Semantic Role Pair & Predicate, SRP&Pred) 特征定义: 若实体 e_1 相对于谓词 V 的语义角色为 R_1 , e_2 相对于 V 的语义角色为 R_2 , 则 $(R_1, s(V), R_2)$ 称为实体对 (e_1, e_2) 相对于谓词 V 的“语义角色对+谓词”组合特征, 其中 $s(V)$ 表示谓词 V 的词干。
- 实体间词语 (Inter-Entity Words, IEW) 特征定义: 2 个实体间的词语的词干称为实体间词语特征。

——《基于语义角色的实体关系抽取》

以上例中识别结果 2 来说明上述 4 个定义。例句首先按照 ProBank 或语义依存关系进行了语义角色的标注。其中, V 代表谓词; A_0 代表施事者; A_1 代表受事者。句子中标记的实体 e_1 为“李克强”, 实体 e_2 为“1955 年 7 月”, 对于上述定义的 4 个特征, 从句子中抽取出的特征值为: “(Arg0, Arg1)”, “出生/V”, “(李克强, 出生, 1955 年 7 月)”, “出生/v 于/p”。

上例中识别结果 2 的语义角色特征如表 10.17 所示。

表 10.17 上例中识别结果 2 的语义角色特征

e1: 李克强	Arg0	e2:1955年7月	Arg1
(e1,e2)	(李克强, 1955年7月)	(R1,R2)	(Arg0, Arg1)
s(V)	出生/V	(R1, s(V),R2)	(李克强, 出生, 1955年7月)
IEW:	出生/v 于/p		

(3) 实体关系抽取训练。按照本文提出的实体关系抽取方案，根据上述一系列特征，利用 SVM 算法构造分类器以判断实体关系类型。

本文使用的实体关系抽取方案具体步骤如下。

- ① 原始语料预处理。对原始语料进行词性标注、句法分析和语义角色标注。
- ② 特征抽取及特征向量构造。对于语料中每条句子中的实体对，先从预处理后的文本中抽取上文描述的特征，然后将抽取出的每个特征值作为实体对的特征向量中的一维，由此构成了实体对的特征向量。
- ③ 构造分类器。用训练语料中实体对的特征向量构造 SVM 分类器。
- ④ 输出分类。利用训练得到的 SVM 分类器判断测试语料中实体对的关系类型。

毛小丽等在《基于语义角色的实体关系抽取》一文中，使用 SemEval-2010 评测任务 8 进行语义角色标注的实体关系抽取实验。在 SemEval-2010 评测任务 8 将实体关系类型分为 9 类，提供的训练语料包含 8 000 个句子，每个句子均标出了 2 个实体及其所属关系类型。在《基于语义角色的实体关系抽取》论文所述的实验中，将 8 000 个句子的前 800 个句子作为测试语料，其余的句子作为训练语料。实验首先对语料进行词性标注、句法分析和语义角色标注等预处理。其中，语义角色标注使用了 CCG SRL (Punyakank et al, 2004) 工具。然后按照常用特征和语义角色特征抽取方法产生特征向量。最后使用 LibSVM 对抽取出的特征向量进行训练分类。论文中的实验分为如下两个实验。实验 1 对本文提出的语义角色特征进行了验证，并与常用特征的实验结果进行了比较。实验 2 使用不同的机器学习方法构造分类器以验证 SVM 在本文提出的关系抽取方案中的有效性。本节仅给出实验 1 各种特征下的对比结果，如表 10.18 所示。

表 10.18 为不同特征选取方案的性能对比

类 别	说 明	P (%)	R (%)	F1
特征集1	常用特征	75.2	80.2	0.776
特征集2	常用特征+R1+R2	74.0	80.5	0.771
特征集3	常用特征+SRP	74.2	80.6	0.773

续表

类 别	说 明	P (%)	R (%)	F1
特征集4	常用特征+ SRP+Pred+SRP&Pred (未提取谓词词干)	76.0	81.4	0.786
特征集5	常用特征+ SRP+Pred+SRP&Pred+IEW (未提取谓词和实体间词语词干)	77.5	84.8	0.810
特征集6	常用特征+ SRP+Pred+SRP&Pred+IEW	78.1	85.4	0.816

如表 10.19 所示，虽然实验使用的是英文语料，但是实验的过程比较细致，比起那种在大规模未经校对的中文语料上的实验结果更能说明问题。得出如下结果。

特征集 1 和特征集 3 的对比实验结果表明，在常用特征基础上只加入语义角色对特征反而降低了关系抽取性能，这是因为任何一个实体的语义角色都是相对于一个谓词而言的，离开了谓词语义角色就失去了本来应该包含的丰富的语义信息。特征集 2 和特征集 3 的实验对比表明，语义角色对整体作为一个特征比实体的每个语义角色分别作为特征更能有效地区分 2 个实体间的关系类型。特征集 3 和特征集 4 的对比实验表明，语义角色对所依赖的谓词能够进一步提高实体关系抽取性能，并且通过特征集 1 和特征集 3 的对比实验结果类推，语义角色对+谓词组合特征对实体关系的分类有一定的意义。特征集 4 和特征集 5 的对比实验结果显示实体间词语特征对区分不同的实体关系有一定的作用，这是因为有些时候句子中 2 个实体不存在语义角色对特征，这时就需要实体间词语特征区分实体间关系。特征集 5 和特征集 6 的对比实验结果表明，抽取出词语的词干对提高分类性能是有好处的。

从特征集 1 和特征集 6 的实验对比结果可以看出，在常用特征基础上加入语义角色特征后系统对于关系分类的 F1 值比没有加入语义角色特征时提高了 4%，在一定程度上提高了实体关系抽取性能。

——《基于语义角色的实体关系抽取》

10.4 结语

本章是本书的最后一章。本章结合前文中所有的研究成果，给出了句子语义解析的完整架构（见图 10.9）。

在完整架构中，汉语自然语言处理分为三个部分：词法分析、句法分析和句义分析。三者之间既相互独立，又相互联系。词法分析中的中文分词是汉语 NLP 中特有的模块，

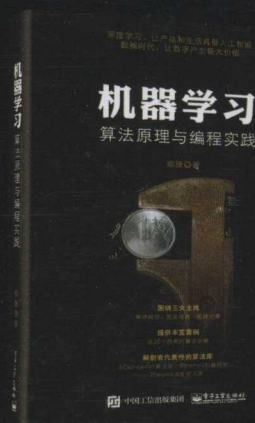
它把连续的汉语字符串分为用空格间隔的词汇列表,使汉语能够像印欧语言一样变为分词连写的形式。这是汉语 NLP 整个流程所需的一种特有的环节。之后像所有的语言处理程序一样,为每个词汇标注出相应的词性;或者标出由一个词或几个词所构成的命名实体;或者标出由一个词或几个词所构成的语义组块。以便为后续处理所用。

句法分析传统上是 NLP 处理的最核心的环节。对它的研究已经经历了半个多世纪,但迄今为止,完全句法解析的 F 值仍旧没有达到实用的要求。这里将句法分析作为句子化繁为简的一个重要环节,在它的基础上,形成两个重要的模块:长句切分和融合、共指消解。虽然本书给出了两个著名的实现,但从总体上讲,共指消解的研究还远未达到实用的程度,长句切分的精度也有待进一步提高。这将会成为下一步研究的重点。

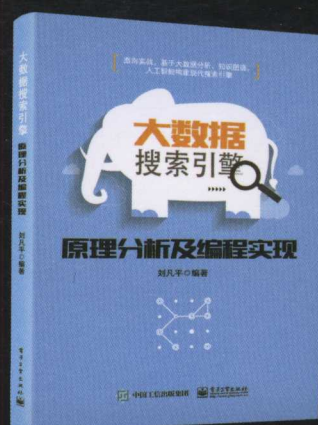
新生的认知语言学从理论上否定了语言的句法分析是走向语义的唯一途径,而提出了一系列新的基于认知的语言学理论,在这些理论基础之上,人们后来发展出了语义角色标注理论作为句子语义分析的基本数据结构。哈工大 LTP 的语义依存模块进一步发展了语义角色标注理论,而形成了:语义角色标注+依存树的整合形式。该形式跨越了句法环节,而从分词处理环节后直接生成,是汉语 NLP 句子分析的一个重要的成果,具有里程碑的意义。汉语语言处理逐渐从理论和实践上摆脱传统的语言学思想,而进入了一个新的阶段。

表 10-18 为不同特征提取方案的精度对比

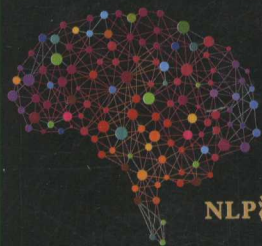
特征集	第四特征	第五特征	第六特征	第七特征
特征集1	第四特征集1-4	第五特征集1-5	第六特征集1-6	第七特征集1-7
特征集2	第四特征集2-4	第五特征集2-5	第六特征集2-6	第七特征集2-7



ISBN: 978-7-121-27367-4
2015年10月出版



ISBN: 978-7-121-29164-7
2016年7月出版



NLP汉语自然语言处理原理与实践

本书作者通过对前人语言学理论和自然语言处理技术的深入梳理，形成了自己对于语义理解，特别是汉语语义理解独特的思考和一整套理论体系，提出了语义理解的系统解决之道。尽管如何才能让计算机理解语义，在学术界还没有定论，但作者系统性的思考和解决思路是非常难能可贵的。本书在内容上保证了理论和技术的平衡，充分展示了作者对于道的思考成果。此书是自然语言处理书籍中的一股新风，可以对语义理解的研究和发展起到积极的推动作用，同时引导自然语言处理领域的研究者，特别是初学者，加强对于语言学的理论的学习，更多地从问题的本源来寻求新的解决思路，而不仅仅满足于在传统解决思路尝试新的技术手段。

——360 算法设计师 贾文杰

作为一本专业研究自然语言处理（NLP）的书籍，本书兼顾了理论、算法和实践三个方面。与以往NLP语言技术类书籍有所不同，本书从NLP历史发展和工程实践两个角度对最新的研究成果进行了详细和深入的探讨，给出了实例代码，并对一些经典的算法源码进行了分析，方便读者学习和模仿。作者具有十多年的NLP研究经验，本书不仅可操作性强，在理论方面也有一些创新，如从认知科学中吸取了诸多相关理论，并把其应用到NLP，是一种难得的创新。

——中国科学院科技战略咨询研究院 宋敦江 博士

随着近些年科技的迅猛发展，人工智能和自然语言处理技术正逐步走向产品化，越来越多的商业公司都尝试着提供语言 and 智能处理设备来理解客户的需求。诸如Facebook和WhatsApp的消息传递平台越来越受欢迎，各种与用户交互的Chatbots通过网站或集成到其他平台，使客户方便地找到他们所关心的问题的答案或相关的资源。这些技术虽然还远未达到成熟的程度，但毕竟使人工智能和NLP的研究成果走出了象牙塔，而进入了人们的生活。作者一直在做NLP认知系统，对于各种智能计算和自然语言处理都有深刻的研究。可以看出本书是作者多年的研究成果，并形成了自己对NLP的认识和见解，是研究和学习NLP相关领域不可多得的书籍。

——约拍啦网络科技有限公司 运营总监 支峥



策划编辑：李 冰 (libing@phei.com.cn)
责任编辑：李 冰
封面设计：朝天世纪

上架建议：计算机-人工智能

ISBN 978-7-121-30765-2



9 787121 307652 >

定价：98.00元